This is a greatly revised and expanded version of "Defeasible reasoning with variable degrees of justification", *Artificial Intelligence* **133** (2001), 233-282.
version of 6/20/02

Defeasible Reasoning with Variable Degrees of Justification II

John L. Pollock
Department of Philosophy
University of Arizona
Tucson, Arizona 85721
pollock@arizona.edu
http://www.u.arizona.edu/~pollock

Abstract

The question addressed in this paper is how the degree of justification of a belief is determined. A conclusion may be supported by several different arguments, the arguments typically being defeasible, and there may also be arguments of varying strengths for defeaters for some of the supporting arguments. What is sought is a way of computing the "on sum" degree of justification of a conclusion in terms of the degrees of justification of all relevant premises and the strengths of all relevant reasons.

I have in the past defended various principles pertaining to this problem. In this paper I reaffirm some of those principles but propose a significantly different final analysis. Specifically, I endorse the weakest link principle for the computation of argument strengths. According to this principle the degree of justification an argument confers on its conclusion in the absence of other relevant arguments is the minimum of the degrees of justification of its premises and the strengths of the reasons employed in the argument. I reaffirm my earlier rejection of the accrual of reasons, according to which two arguments for a conclusion can result in a higher degree of justification than either argument by itself. This paper diverges from my earlier theory mainly in its treatment of defeaters. First, it argues that defeaters that are too weak to defeat an inference outright may still diminish the strength of the conclusion. Second, in the past I have also denied that multiple defeaters can result in the defeat of an argument that is not defeated by any of the defeaters individually. In this paper I urge that there are compelling examples that support a limited version of this "collaborative" defeat.

The need to accommodate diminishers and collaborative defeat has important consequences for the computation of degrees of justification. The paper proposes a characterization of degrees of justification that captures the various principles endorsed and constructs an algorithm for computing them. It then goes on to derive an equivalent formulation of the theory that is more amenable to implementation.

Keywords: defeasible, defeat, justification, reasoning, nonmonotonic

---

This work was supported by NSF grants nos. IRI-9634106 and IIS-0080888.

1. Introduction

I have argued at length elsewhere that a rational agent operating in a complex environment must reason about its environment defeasibly.[1] For example, perceptual input is not always accurate, so an agent forming beliefs about its environment on the basis of its sensors must be prepared to withdraw such beliefs in the face of further information. A sophisticated agent should be able to discover generalizations about its environment by reasoning inductively, but inductive reasoning is defeasible—new evidence may overturn earlier generalizations. Because perception only enables an agent to monitor small parts of its environment at any one time, in order to build a coherent world model the agent must combine conclusions drawn on the basis of different perceptual experiences occurring at different times. But that requires a defeasible assumption that the world does not change too rapidly, so that what was perceived a moment ago is still true. The ability to maneuver through a rich environment requires an agent to be able to reason about the causal consequences of both its own actions and other events that it observes. That requires a solution to the frame problem, and it is generally agreed that any such solution will require defeasible reasoning.[2] I have also argued that planning with incomplete information requires a defeasible assumption that different plans do not destructively interfere with each other.[3] Although most of these "theoretical" observations are fairly obvious, they have not had much impact on the actual practice of AI, because for the most part people have not tried to build autonomous agents of sufficient sophistication to encounter these problems. However, that is changing and we are getting close to the point where we can construct practical agents that will not work satisfactorily unless their designers take these observations to heart.

The OSCAR architecture for rational agents is based upon a general theory of defeasible reasoning. OSCAR implements the system of defeasible reasoning described in Pollock (1995).[4] That system in turn derives from thirty years of theorizing in philosophical epistemology. The basic idea is that the agent constructs arguments using both deductive and defeasible reason-schemes (inference-schemes). The conclusions of some of these arguments may constitute defeaters for steps of some of the other arguments. Given the set of interacting arguments that represent the agent's epistemological state at a given time, an algorithm is run to compute degrees of justification, determining which arguments are undefeated and the level of support they provide their conclusions. What the agent should believe at any particular time are the conclusions of the undefeated arguments. The hard part of a theory of defeasible reasoning is to give an account of which arguments are undefeated. This is a topic I have addressed before (my 1987, 1994, 1995), but some of the considerations adduced later in this paper have convinced me of the need to revisit it.

My analysis will turn on the taxonomy of defeaters that I introduced in my (1970) and (1974) and that has been endorsed by most subsequent work on defeasible reasoning (see Prakken and

---

[1] The argument spans three decades. My most recent papers in this vein is Pollock (1998) and (1998b), but see also Pollock (1974), (1987), (1990), (1995), and Pollock and Cruz (1999).

[2] I have proposed and implemented a solution to the frame problem in my (1998).

[3] A system of planning based upon this observation was described in my (1998b), and has been implemented in OSCAR.

[4] For a presentation of OSCAR and its current capabilities, see the OSCAR web page at http://oscarhome.soc-sci.edu/ftp/OSCAR-web-page/OSCAR.htm.

Vreeswijk 2002 and Chesñevar, Maguitman, and Loui 2000). According to this taxonomy, there are two importantly different kinds of defeaters. Where P is a defeasible reason for Q, R is a *rebutting defeater* iff R is a reason for denying Q. All work on nonmonotonic logic and defeasible reasoning has recognized the existence of rebutting defeaters, but there are other defeaters as well. For instance, suppose x looks red to me, but I know that x is illuminated by red lights and red lights can make objects look red when they are not. Knowing this defeats the defeasible reason, but it is not a reason for thinking that x is *not* red. After all, red objects look red in red light too. This is an *undercutting defeater*. Undercutting defeaters attack the *connection* between the reason and the conclusion rather than attacking the conclusion directly. For example, an undercutting defeater for the inference from x's looking red to x's being red attacks the connection between "x looks red to me" and "x is red", giving us a reason for doubting that x wouldn't look red unless it were red. I will symbolize the negation of "P wouldn't be true unless Q were true" as "$P \otimes Q$". A shorthand reading is "P does not guarantee Q". If Γ (a set of propositions) is a defeasible reason for P, then where $\Pi\Gamma$ is the conjunction of the members of Γ, any reason for believing "$\Pi\Gamma \otimes P$" is a defeater. Thus I propose to characterize undercutting defeaters as follows:

> If Γ is a defeasible reason for P, an *undercutting defeater* for Γ as a defeasible reason for P is any reason for believing "$(\Pi\Gamma \otimes P)$".

Are there any defeaters other than rebutting and undercutting defeaters? A number of authors have advocated *specificity defeaters* (e.g., Touretzky 1984, Poole 1988, Simari and Loui 1992). These have been formulated differently by different authors, but the general idea is that if two arguments lead to conflicting conclusions but one argument is based upon more information than the other then the "more informed" argument defeats the "less informed" one.

A phenomenon like this is common in several different kinds of probabilistic reasoning. To illustrate, consider the statistical syllogism. The statistical syllogism can be formulated as follows (see my 1990):

(SS) If $r > 0.5$, then "Fc & $\text{prob}(G/F) \geq r$" is a defeasible reason for believing "Gc", the strength of the reason depending upon the value of r.

When reasoning in accordance with (SS), there is a kind of "total evidence requirement" according to which we should make our inference on the basis of the most comprehensive facts regarding which we know the requisite probabilities. This can be accommodated by endorsing the following undercutting defeater for (SS):

> "Hc & $\text{prob}(G/F\&H) \neq \text{prob}(G/F)$" is an undercutting defeater for (SS).

I refer to these as *subproperty defeaters.*[5]

Early work in AI on defeasible reasoning tended to concentrate on examples that were

[5] I first pointed out the need for subproperty defeaters in my (1983). Touretzky (1984) subsequently introduced similar defeaters for use in defeasible inheritance hierarchies.

instances of the statistical syllogism (e.g., the venerable "Tweety" arguments), and this led people to suppose that something like subproperty defeat was operative throughout defeasible reasoning. However, I do not see any reason to believe that. There are several specific kinds of defeasible reasoning that are subject to total-evidence requirements. The statistical syllogism is one. Direct inference (discussed in section nine below) is another. Don Nute has pointed out to me that various kinds of legal reasoning and deontic reasoning are subject to a similar requirement. These can all be accommodated by acknowledging similar subproperty defeaters (which are undercutting defeaters) for the defeasible inference-schemes involved in the reasoning. To the best of my knowledge, there has never been an intuitive example of specificity defeat presented anywhere in the literature that is not an example of the operation of the total-evidence requirement in one of these special varieties of defeasible inference, and the latter are all instances of undercutting defeat. Accordingly, I will assume that undercutting defeaters and rebutting defeaters are the only possible kinds of defeaters.

I have defended the preceding remarks at length in numerous publications over the past 30 years, so for the purposes of this paper I will regard them as ancient history and take them for granted without further discussion. They are not the topic of this paper. This paper is about how to compute defeat statuses. The literature on defeasible and nonmonotonic reasoning contains numerous proposals for how to do this.[6] The current version of OSCAR computes defeat statuses in the manner described in my (1994) and (1995).[7] If we ignore the fact that some arguments provide stronger support for their conclusions than other arguments, we can describe OSCAR's defeat status computation as follows. We collect arguments into an *inference-graph*, where the nodes represent the conclusions of arguments, *support-links* tie nodes to the nodes from which they are inferred, and *defeat-links* indicate defeat relations between nodes. These links relate their *roots* to their *targets*. The root of a defeat-link is a single node, and the root of a support-link is a set of nodes. The analysis is somewhat simpler if we construct the inference-graph in such a way that when the same conclusion is supported by two or more arguments, it is represented by a separate node for each argument. For example, consider the inference-graph diagrammed in figure one, which represents two different arguments for (*P*&*Q*) given the premises, *P*, *Q*, *A*, and (*Q* → (*P*&*Q*)). The nodes of such an inference-graph represent arguments rather than just representing their conclusions. In such an inference-graph, a node has at most one support-link. When it is unambiguous to do so, I will refer to the nodes in terms of the conclusions they encode.

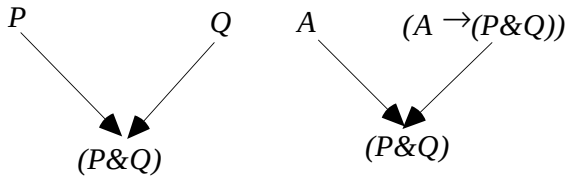


Figure 1. An inference-graph

[6] Two good surveys are Prakken and Vreeswijk (2002) and Chesñevar, Maguitman, and Loui (2000).

[7] For comparison with default logic and circumscription, see my (1995), chapter three. For comparison with more recent systems of defeasible argumentation, see Prakken and Vreeswijk (2002).

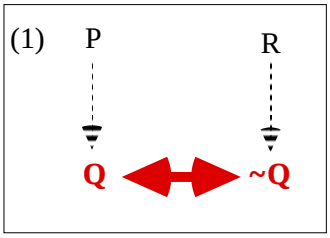
The *node-basis* of a node is theset of roots of its support-links (if it has any), i.e., the set of nodes from which the node is inferred in a single step. If a node has no support-link (i.e., it is a premise) then the node-basis is empty. The *node-defeaters* are the roots of the defeat-links having the node as their target. We define:

A node of the inference-graph is *initial* iff its node-basis and list of node-defeaters is empty.

The *defeat status* of a node is either "defeated" or "undefeated". It is initially tempting to try to characterize defeat statuses recursively using the following three rules:

(1) Initial nodes are undefeated.
(2) A node is undefeated if all the members of its node-basis are undefeated and all node-defeaters are defeated.
(3) A node is defeated if either some member of its node-basis is defeated or some node-defeater is undefeated.

However, this recursion turns out to be ungrounded because we can have nodes of an inference-graph that defeat each other, as in inference-graph (1), where dashed arrows indicate defeasible inferences and heavy arrows indicate defeat-links. In computing defeat statuses in inference-graph (1), we cannot proceed recursively using rules (1)–(3), because that would require us to know the defeat status of *Q* before computing that of ~*Q*, and also to know the defeat status of ~*Q* before computing that of *Q*. The problem is more generally that a node *P* can have an inference/defeat-descendant that is a defeater of *P*, where an inference/defeat-descendant of a node is any node that can be reached from the first node by following support-links and defeat-links. I will say that a node is *P-dependent* iff it is an inference/defeat-descendant of a node *P*. So the recursion is blocked in inference-graph (1) by there being *Q*-dependent defeaters of *Q* and ~*Q*-dependent defeaters of ~*Q*.



Inference-graph (1) is a case of "collective defeat". For example, let *P* be "Jones says that it is raining", *R* be "Smith says that it is not raining", and *Q* be "It is raining". Given *P* and *Q*, and supposing you regard Smith and Jones as equally reliable, what should you believe about the weather? It seems clear that you should withhold belief, accepting neither. In other words, both *Q* and ~*Q* should be defeated. This constitutes a counter-example to rule (2). So not only do rules (1)–(3) not provide a recursive characterization of defeat statuses — they are not even true. The failure of these rules to provide a recursive characterization of defeat statuses suggests that no such characterization is possible, and that in turn suggested to me (in my 1994, 1995) that rules

(1)–(3) might be used to characterize defeat statuses in another way. Reiter's (1980) default logic proceeded in terms of multiple "extensions", and "skeptical default logic" characterizes a conclusion as following nonmonotonically from a set of premises and defeasible inference-schemes iff it is true in every extension. The currently popular stable model semantics (Dung 1995) is derived from this approach. There are simple examples showing that these semantics are inadequate for the general defeasible reasoning of epistemic agents (see section two), but the idea of having multiple extensions suggested to me that rules (1)–(3) might be used to characterize multiple "status assignments". On this approach, a status assignment is an assignment of defeat statuses to the nodes of the inference-graph in accordance with three simple rules:

> An assignment σ of "defeated" and "undefeated" to a subset of the nodes of an inference-graph is a *partial status assignment* iff:
> 1. σ assigns "undefeated" to any initial node;
> 2. σ assigns "undefeated" to a node α iff σ assigns "undefeated" to all the members of the node-basis of α and all node-defeaters of α are assigned "defeated"; and
> 3. σ assigns "defeated" to a node α iff either some member of the node-basis of α is assigned "defeated", or some node-defeater of α is assigned "undefeated".
>
> σ is a *status assignment* iff σ is a partial status assignment and σ is not properly contained in any other partial status assignment.

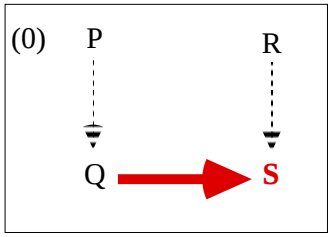
My proposal was then:

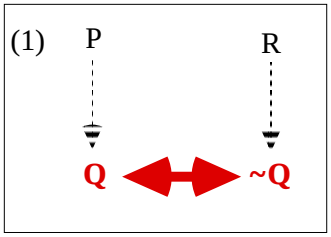
> A node is undefeated iff every status assignment assigns "undefeated" to it; otherwise it is defeated.
>
> Belief in *P* is justified for an agent iff *P* is encoded by an undefeated node of the inference-graph representing the agent's current epistemological state.

2. Examples

The ultimate test of a semantics is that it validates the intuitively right reasoning. It is human intuitions about correct reasoning that we want to capture. (See Pollock & Cruz (1999), chapter six, for further discussion of this.) With this in mind, it will be useful to have a number of examples of inference-graphs to test the analysis I will propound below. I will assume throughout this section that all initial nodes (premises) have the same degree of justification, and all reason-schemes have the same reason-strength.

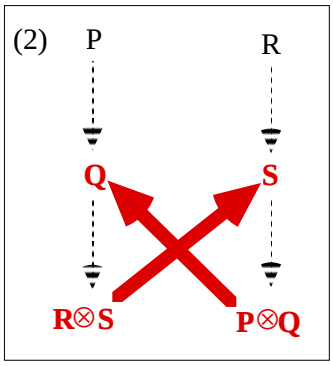
The simplest case is inference-graph (0). Presumably, any semantics for defeasible reasoning will yield the result that *S* is defeated, and *P*, *Q*, and *R* are undefeated.

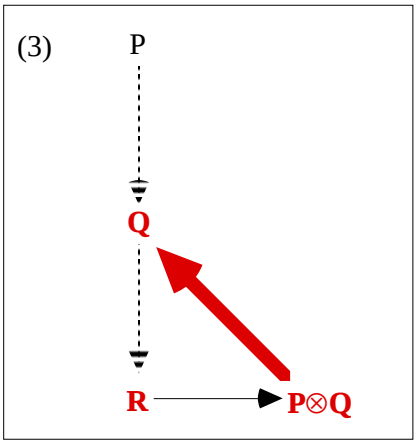




Inference-graph (1) illustrates "collective defeat". It was discussed above, and again all semantics for defeasible reasoning yield the result that *Q* and ~*Q* are defeated, but *P* and *R* are undefeated.

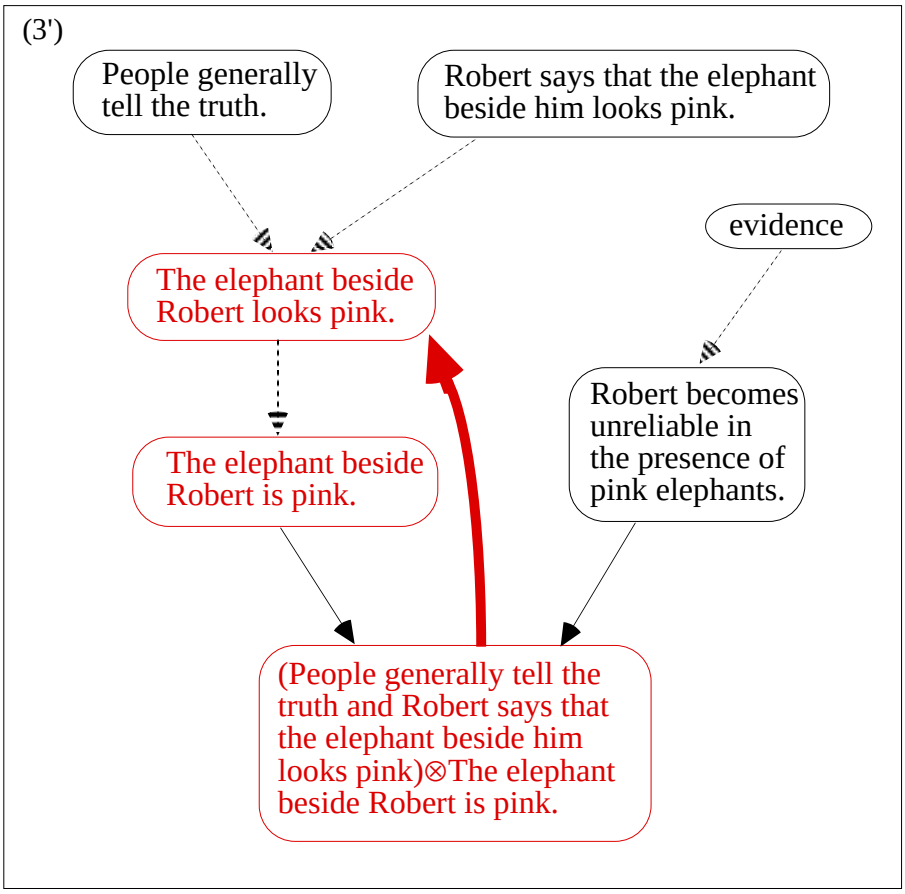
Inference-graph (2) is another instance of collective defeat, differing from inference-graph (1) in that the defeaters are undercutting defeaters rather than rebutting defeaters. The result should be that *Q*, *S*, *R*⊗*S*, and *P*⊗*Q* are defeated, and *P* and *R* are undefeated. For an example, let *P* = "Jones says Smith is untrustworthy", *R* = "Smith says Jones is untrustworthy", *Q* = "Smith is untrustworthy", *S* = "Jones is untrustworthy". The semantics of section one produces two status assignments, one in which *Q* and *R*⊗*S* are assigned "defeated" and all other nodes are assigned "undefeated", and one in which *S* and *P*⊗*Q* are assigned "defeated" and all other nodes are assigned "undefeated".

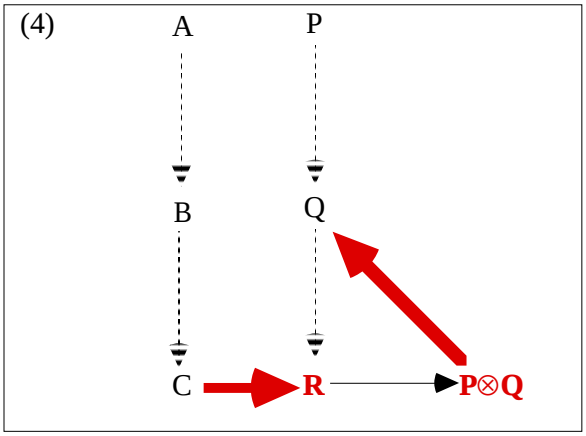




Inference-graph (3) is a simple example of a "self-defeating" argument. A partial status assignment must assign "undefeated" to *P*. If it assigned "undefeated" to *Q* then it would assign "undefeated" to *R* and *P*⊗*Q*, in which case it would have to assign "defeated" to *Q*. So it cannot assign "undefeated" to *Q*. If it assigned "defeated" to *Q* it would have to assign "defeated" to *R* and *P*⊗*Q*, in which case it would have to assign "undefeated" to *Q*. So that is not possible either. Thus a partial status assignment cannot assign anything to *Q*, *R*, and *P*⊗*Q*. Hence there is only one status assignment (i.e., maximal partial status assignment). Accordingly, *P* is undefeated and the other nodes are defeated. An intuitive example having approximately the same form is shown in inference-graph (3'). Inference-graphs (3) and (3') constitute intuitive counterexamples to default logic (Reiter 1980) and the stable model semantics (Dung 1995) because there are no extensions. Hence on those semantics, *P* has the same status as *Q*, *R*, and *P*⊗*Q*. It is perhaps more obvious that this is a problem for those semantics if we imagine this self-defeating argument being embedded in a larger inference-graph containing a number of otherwise perfectly ordinary

arguments. On these semantics, all of the nodes in all of the arguments would have to have the same status, because there would still be no extensions. But surely the presence of the self-defeating argument should not have the effect of defeating all other (unrelated) arguments.

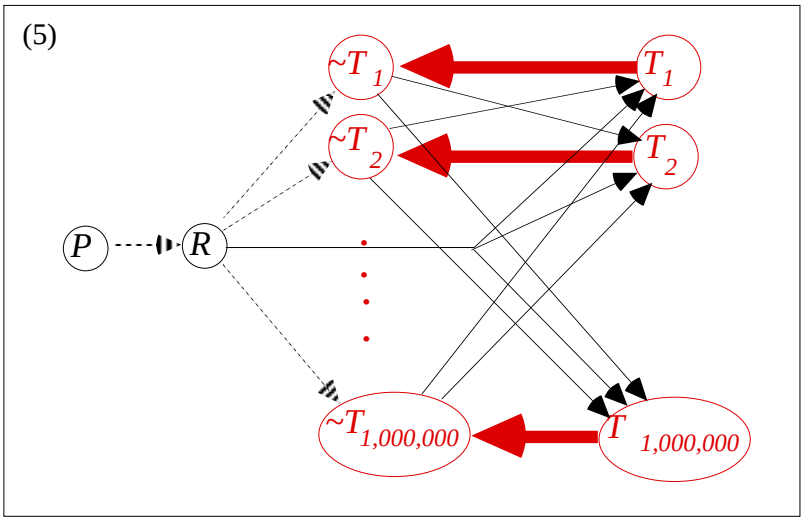




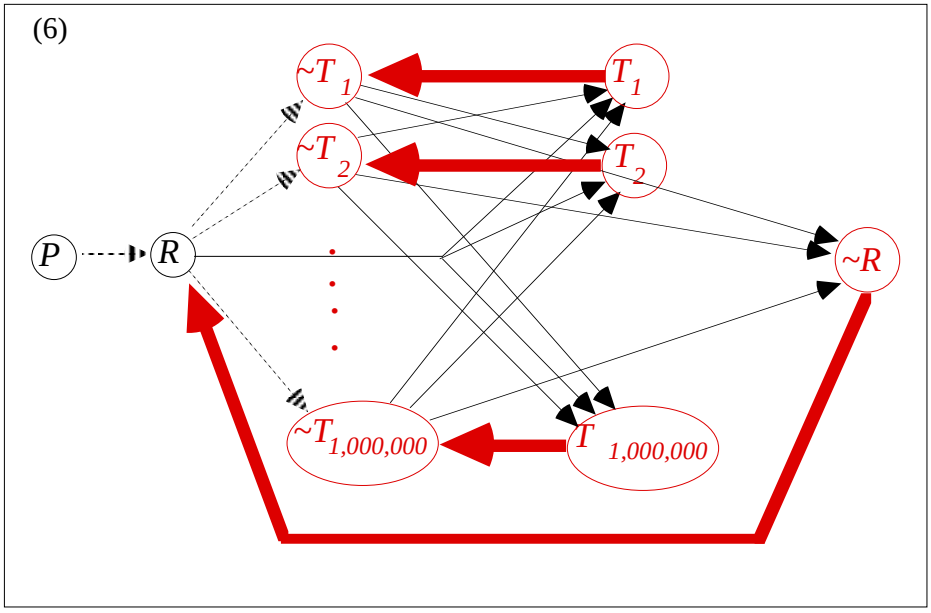
Inference-graph (4), illustrates that self-defeat can be "cancelled" by external defeat. Here *R* is defeated by *C*, so *P*⊗*Q* is defeated and *Q* is undefeated. Accordingly, there is just one status assignment, assigning "undefeated" to *A*, *B*, *C*, *P*, and *Q*, and "defeated" to *R* and *P*⊗*Q*.

Inference-graph (5) illustrates the so-called "lottery paradox" (Kyburg 1961). Here *P* reports a

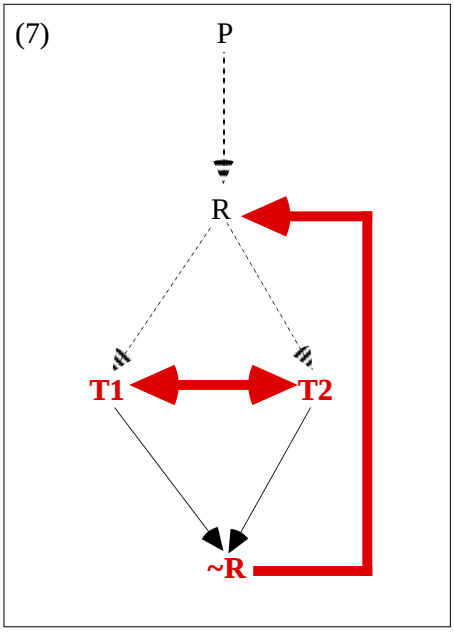
description (e.g., a newspaper report) of a fair lottery with one million tickets. *P* constitutes a defeasible reason for *R*, which is the description. In such a lottery, each ticket has a probability of one in a million of being drawn, so for each *i*, the statistical syllogism gives us a reason for believing $\sim T_i$ ("ticket *i* will not be drawn"). The supposed paradox is that although we thusly have a reason for believing of each ticket that it will not be drawn, we can also infer on the basis of *R* that some ticket will be drawn. Of course, this is not really a paradox, because the inferences are defeasible and this is a case of collective defeat. This results from the fact that for each *i*, we can infer T_i from the description *R* (which entails that some ticket will be drawn) and the conclusions that none of the other tickets will be drawn. This gives us a defeating argument for the defeasible argument to the conclusion that $\sim T_i$, as diagrammed in inference-graph (5). The result is that for each *i*, there is a status assignment on which $\sim T_i$ is assigned "defeated" and the other $\sim T_j$'s are all assigned "undefeated", and hence none of them are assigned "undefeated" in every status assignment.



I believe that all (skeptical) semantics for defeasible reasoning get the lottery paradox right. A more interesting example is the "lottery paradox *paradox*", diagrammed in inference-graph (6). This results from the observation that because *R* entails that some ticket will be drawn, from the collection of conclusions of the form $\sim T_i$ we can infer $\sim R$, and that is a defeater for the defeasible inference from *P* to *R*. This is another kind of self-defeating argument. Clearly, the inferences in the lottery paradox should not lead us to disbelieve the newspaper's description of the lottery, so *R* should be undefeated. Circumscription (McCarthy 1986), in its simple non-prioritized form, gets this example wrong, because one way of minimizing abnormalities would be to block the inference from *P* to *R*. My own early analysis (Pollock 1987) also gets this wrong. This was the example that led me to the analysis of section one. That analysis gets this right. We still have the same status assignments as in inference-graph (5), and $\sim R$ is defeated in all of them because it is inferred from the entire set of $\sim T_i$'s, and one of those is defeated in every status assignment.



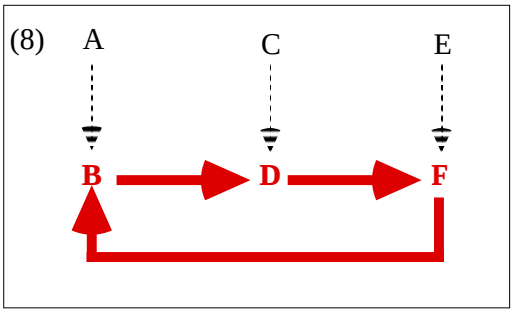
It will be convenient to have a simpler example of an inference-graph with the same general structure as the lottery paradox paradox. For that purpose we can use inference-graph (7). Here *P* and *R* should be undefeated, but *T1*, *T2*, and **~***R* should be defeated.



A final example that creates interesting problems involves "odd-length defeat cycles", as in inference-graph (8). For example, let *A* = "Jones says that Smith is unreliable", *B* = "Smith is unreliable", *C* = "Smith says that Robinson is unreliable", *D* = "Robinson is unreliable", *E* = "Robinson says that Jones is unreliable", *F* = "Jones is unreliable". Intuitively, this should be another case of collective defeat, with *A*, *C*, and *E* being undefeated and *B*, *D*, and *F* being defeated. The semantics of section one does yield this result, but it does it in a peculiar way. *A*, *C*, and *E* must be assigned "undefeated", but there is no consistent way to assign defeat statuses

of *B*, *D*, and *F*. Accordingly, there is only one status assignment (maximal partial status assignment), and it leaves *B*, *D*, and *F* unassigned. We get the right answer, but it seems puzzling that we get it in a different way than we do for even-length defeat cycles like that in inference-graph (1). This difference has always bothered me.

(8)



3. Varying Degrees of Justification

The preceding account of defeat status assumes that all arguments support their conclusions equally strongly. However, this assumption is unrealistic. For example, increasing the degrees of justification of the premises of an argument may increase the degree of justification of the conclusion, and increasing the strengths of the reasons employed in the argument may increase the degree of justification of the conclusion. This phenomenon has been ignored in most AI work on defeasible and nonmonotonic reasoning, but it is of considerable importance in applications of such reasoning. For example, in Pollock (1998) I discussed temporal projection, wherein it is assumed defeasibly that if something is true at one time it is still true at a later time. This is, in effect, a defeasible assumption that fluents are stable, tending not to change truth values as time passes. The stability of a fluent is measured by the probability ρ that if it is true at time *t* then it is still true at time *t*+1. More generally, if it is true at time *t*, then the probability of its being true at $t+\Delta t$ is $\rho^{\Delta t}$. So the strength of the defeasible expectation supported by temporal projection is a monotonic decreasing function of the time interval. This can be captured in a system of defeasible reasoning by employing a reasoning scheme of the following sort:

> "*P*-at-t" is a defeasible reason for believing "*P*-at-($t+\Delta t$)", the strength of the reason being a monotonic decreasing function of Δt.[8]

The decreasing strength is important in understanding perceptual updating, wherein on the basis of new perceptual experience the agent overrides temporal projection and concludes that the fluent has changed truth value. Perception is not infallible, so perception should provide only a defeasible reason for believing that the environment is as represented by the percept.[9] Suppose an object looks red at one time t_1 and blue at a later time t_2. The agent should assume

[8] The reason-schema proposed in Pollock (1998) involves some additional qualifications, but they are not relevant to the present discussion.

[9] This is discussed in detail in Pollock (1998).

defeasibly that the object is initially red, but should also conclude defeasibly that it changes color later and is blue at t_2. The object's being red at t_1 provides a defeasible reason for expecting it to be red at t_2, and its looking blue at t_2 provides a defeasible reason for thinking it blue and hence not red at t_2. If these reasons were of the same strength, there would be no basis for preferring one conclusion to the other and the agent would be unable to draw a justified conclusion about the color of the object at t_2. The situation is resolved by noting that the reason for thinking the object is still red at t_2 is weaker than the reason for thinking it was red at t_1, and hence weaker than the reason for thinking the object is blue (and so not red) at t_2. Because the agent has a stronger reason for thinking the object is blue at t_2 than for thinking it is red at t_2, that becomes the justified conclusion and the agent is able to conclude that the object has changed color.

The preceding example illustrates the importance of incorporating an account of degrees of justification into a system of defeasible reasoning. There are many other examples illustrating the same point. For instance, in the statistical syllogism the strength of the reason is a function of the probability employed. Autonomous agents capable of engaging in sophisticated defeasible reasoning must accommodate varying degrees of justification. The question addressed in this paper is how the degree of justification of a conclusion should be determined. A conclusion may be supported by several different arguments. The arguments are typically defeasible, and there may also be arguments of varying strengths for defeaters for some of the supporting arguments. What is sought is a way of computing the "on sum" degree of justification of a conclusion. It is clear that three variables, at least, are involved in determining degrees of justification. The reason-strengths of the reason-schemes employed in the argument are relevant. The degrees of justification of the premises are relevant. And the degrees of justification of any defeaters for defeasible steps of the argument are relevant. Other variables might be relevant as well. I am going to assume that reason-strengths and degrees of justification are measurable as non-negative extended real numbers (i.e., the non-negative reals together with ∞). The justification for this assumption will be provided in section ten.

4. Argument-Strengths

For the sake of completeness, this section and the next repeat arguments given in my (1995). Let us begin by looking at arguments for which we have no arguments supporting defeaters. Let the *strength of an argument* be the degree of justification it would confer on its conclusion under those circumstances. A common and seductive view would have it that argument strength can be modeled by the probability calculus. On this view, the strength a conclusion gains from the premises can be computed in accordance with the probability calculus from the strength of the reason (a conditional probability) and the probabilities of the premises. I, and many other authors, have argued against this view at length, but it has a remarkable ability to keep attracting new converts.

There are a number of familiar arguments against the probabilistic model. The simplest argument proceeds by observing that the probabilistic model would make it impossible to be justified in believing a conclusion on the basis of a deductive argument from numerous uncertain premises. This is because as you conjoin premises, if degrees of support work like probabilities, the degree of support decreases. Suppose you have 100 independent premises, each highly

probable, having, say, probability .99. According to the probability calculus, the probability of the conjunction will be only .37, so we could never be justified in using these 100 premises conjointly in drawing a conclusion. But this flies in the face of human practice. For example, an engineer building a bridge will not hesitate to make use of one hundred independent measurements to compute (deduce) the correct size for a girder. I have discussed this issue at length elsewhere (Pollock 1987, 1995, Pollock and Cruz 1999), so in this paper I am just going to assume that deductive arguments provide one way of arriving at new justified conclusions on the basis of earlier ones. A corollary is that the probabilistic model is wrong.[10]

If deductive reasoning automatically carries us from justified conclusions to justified conclusions, then the degree of support a deductive argument confers on its conclusion cannot decrease as the number of premises increases. The degree of justification for the conclusion must be no less than that of the most weakly justified premise. This is the *Weakest Link Principle for Deductive Arguments*, according to which a deductive argument is as good as its weakest link. More precisely:

> The argument strength of a deductive argument is the minimum of the degrees of justification of its premises.

This formulation of the weakest link principle applies only to deductive arguments, but we can use it to obtain an analogous principle for defeasible arguments. If P is a defeasible reason for Q, then we can use conditionalization to construct a simple defeasible argument for the conclusion $(P \rightarrow Q)$, and this argument turns upon no premises:

Suppose P
Then (defeasibly) Q.

Therefore, $(P \rightarrow Q)$.

As this argument has no premises, the degree of support of its conclusion should be a function of nothing but the strength of the defeasible reason. The next thing to notice is that any defeasible argument can be reformulated so that defeasible reasons are only used in subarguments of this form, and then all subsequent steps of reasoning are deductive. The conclusion of the defeasible argument is thus a deductive consequence of the premises together with a number of conditionals justified in this way. By the weakest link principle for deductive arguments, the degree of support of the conclusion should then be the minimum of (1) the degrees of justification of the premises used in the argument and (2) the strengths of the defeasible reasons:

> The argument strength of a defeasible argument is the minimum of the strengths of the defeasible reasons employed in it and the degrees of justification of its premises.

This is the general *Weakest Link Principle*. The problem of computing argument strengths is thus computationally simple.

---

[10] The same objection can be leveled against the Dempster-Shafer theory (Dempster 1968, Shafer 1976).

5. The Accrual of Reasons

If we have two independent reasons for a conclusion, does that make the conclusion more justified than if we had just one? It is natural to suppose that it does,[11] but upon closer inspection that becomes unclear. Cases that seem initially to illustrate such accrual of justification appear upon reflection to be better construed as cases of having a single reason that subsumes the two separate reasons but is not equivalent to their conjunction. For instance, if Brown tells me that the president of Fredonia has been assassinated, that gives me a reason for believing it; and if Smith tells me that the president of Fredonia has been assassinated, that also gives me a reason for believing it. Surely, if they both tell me the same thing, that gives me a better reason for believing it. However, there are considerations indicating that my reason in the latter case is not simply the conjunction of the two reasons I have in the former cases. Reasoning based upon testimony is a straightforward instance of the statistical syllogism. We know that people tend to tell the truth, and so when someone tells us something, that gives us a defeasible reason for believing it. This turns upon the following probability being reasonably high:

(1) prob(*P* is true / *S* asserts *P*).

Given that this probability is high, I have a defeasible reason for believing that the president of Fredonia has been assassinated if Brown tells me that the president of Fredonia has been assassinated.

In the discussion of the weakest link principle, I urged that argument strengths do not conform to the probability calculus. However, that must be clearly distinguished from the question of whether probabilities license defeasible inferences. In fact, I think that a large proportion of our defeasible inferences are based upon probabilities. Such inferences proceed in terms of the statistical syllogism, described in section one and formulated as principle (SS). When we have the concurring testimony of two people, our degree of justification is not somehow computed by applying a predetermined function to probability (1). Instead, it is based upon the quite distinct probability

(2) prob(*P* is true / S_1 asserts *P* and S_2 asserts *P* and $S_1 \neq S_2$).

The relationship between (1) and (2) depends upon contingent facts about the linguistic community. We might have one community in which speakers tend to make assertions completely independently of one another, in which case (2) > (1); and we might have another community in which speakers tend to confirm each other's statements only when they are fabrications, in which case (2) < (1). Clearly our degree of justification for believing *P* will be different in the two linguistic communities. It will depend upon the value of (2), rather than being some function of (1).

It is important to distinguish *epistemic reasoning* — reasoning about what to believe — from *practical reasoning* — reasoning about what actions to perform. These two kinds of reasoning

---

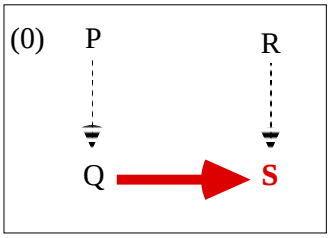
[11] See, for example, Verheij (1996).

have quite different logical properties, as is illustrated at length in Pollock and Cruz (1999). Something like the accrual of reasons seems to hold for practical reasoning. Normally, two independent reasons for choosing a particular action provide a stronger justification for choosing it than either reason by itself. But we cannot conclude from this that the same thing is true of epistemic reasoning.

All examples I have considered that seem initially to illustrate the accrual of reasons in epistemic reasoning turn out in the end to have this same form. They are all cases in which we can estimate probabilities analogous to (2) and make our inferences on the basis of the statistical syllogism rather than on the basis of the original reasons. Accordingly, I doubt that epistemic reasons do accrue. If we have two separate undefeated arguments for a conclusion, the degree of justification for the conclusion is the maximum of the strengths of the two arguments. This will be my assumption.

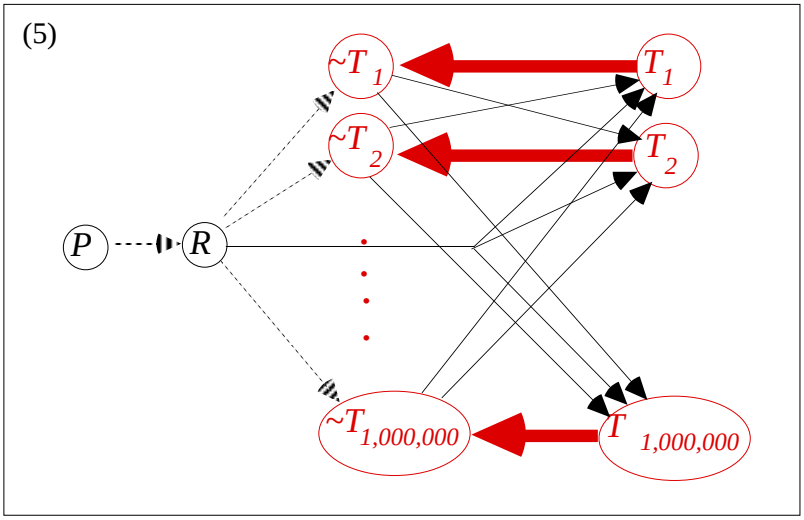
6. The Influence of Defeaters

Thus far I have considered how reason-strengths and the degrees of justification of premises affect the degree of justification of a conclusion. The third variable we must consider is the presence of arguments supporting defeaters. Suppose we have only two arguments to consider, and the conclusion of one of them is a defeater for the final step of the other, as diagrammed in inference-graph (0). How should this affect the degree of justification of *S*?



It seems clear that if the argument strength of the left argument (for *Q*) is as great as that of the right argument (for *S*), then the degree of justification of *S* should be 0. But what if the argument strength of the left argument is less than that of the right argument? In my (1995), I maintained that defeat was an all-or-nothing matter, and hence weaker defeaters leave arguments unaffected. In the scenario just described, this has the consequence that the degree of justification of *S* is the same as the argument strength of the right argument. However, there are some examples that now convince me that this is incorrect. The simplest examples have to do with biased lotteries. To see how these examples work, recall the earlier analysis of reasoning about fair lotteries. Consider a fair lottery consisting of 1 million tickets, and suppose it is known that one and only one ticket will win. Observing that the probability is only .000001 of any particular ticket being drawn given that it is a ticket in the lottery, it seems initially reasonable to employ the statistical syllogism and accept the conclusion regarding any particular ticket that it will not be drawn. This reasoning is completely general and applies to each ticket. However, these conclusions conflict jointly with something else we are justified in believing, viz., that some ticket will be drawn. We cannot be justified in believing each member of an explicitly contradictory

set of propositions, and we have no way to choose between them, so it follows intuitively that we are not justified in believing of any ticket that Jones did not hold that ticket. The formal reconstruction of this reasoning proceeds by observing that this is a case of collective defeat. For each *n*, the statistical syllogism provides a defeasible reason for believing "$\sim T_n$". But for each *k*, we have an equally strong defeasible reason for believing each "$\sim T_k$". We know that some ticket will be drawn. Thus we can construct the counterargument diagramed in inference-graph (5) for the conclusion that "T_n" is true. Our reason for believing each "$\sim T_k$" is as good as our reason for believing "$\sim T_n$", so we have as strong a reason for "T_n" as for "$\sim T_n$". Hence our defeasible reason for the latter is defeated and we are not justified in believing "$\sim T_n$".



Next, consider lottery 2, which is a biased lottery consisting of just ten tickets. The probability of ticket 1 being drawn is .000001, and the probability of any other ticket being drawn is .111111. It is useful to diagram these probabilities as in figure 2. In *this* lottery, it seems reasonable to infer that ticket 1 will not be drawn, because the probability of any other ticket being the drawn is more than 100,000 times greater. This can be justified as follows. As before, we have a defeasible reason for believing "$\sim T_n$", for each *n*. But these reasons are no longer of equal strength. Because ticket 1 is much less likely to be drawn than any other ticket, we have a much stronger reason for believing that ticket 1 will not be drawn. As before, for $n > 1$, we have the counterargument diagramed in inference-graph (5) for "T_n", and that provides as good a reason for believing "T_n" as we have for believing "$\sim T_n$". Thus the defeasible reason for "$\sim T_n$" is defeated. But we do not have as good a reason for believing "T_1" as we do for believing "$\sim T_1$". An argument is only as good as its weakest link, and the counterargument for "T_1" employs the defeasible reasons for "$\sim T_n$" for $n > 1$. These reasons are based upon lower probabilities (of value .888889) and hence are not as strong as the defeasible reason for "$\sim T_1$" (based upon a probability of value .999999). Thus, although we have a reason for believing "T_1", we have a better reason for believing "$\sim T_1$", and so on sum we are justified in believing the latter.

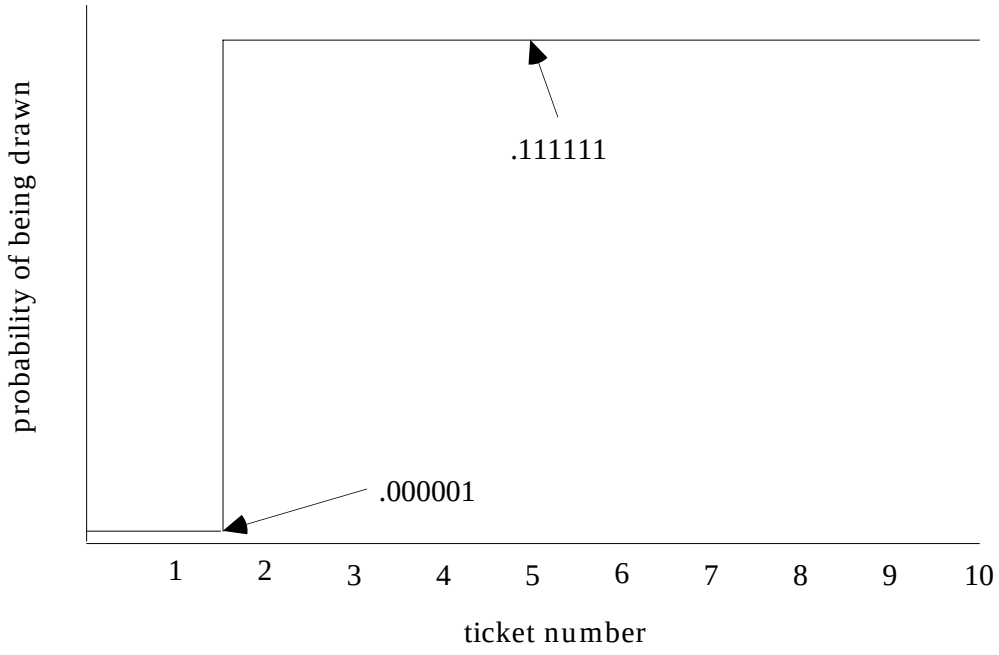


Figure 2. Lottery 2.

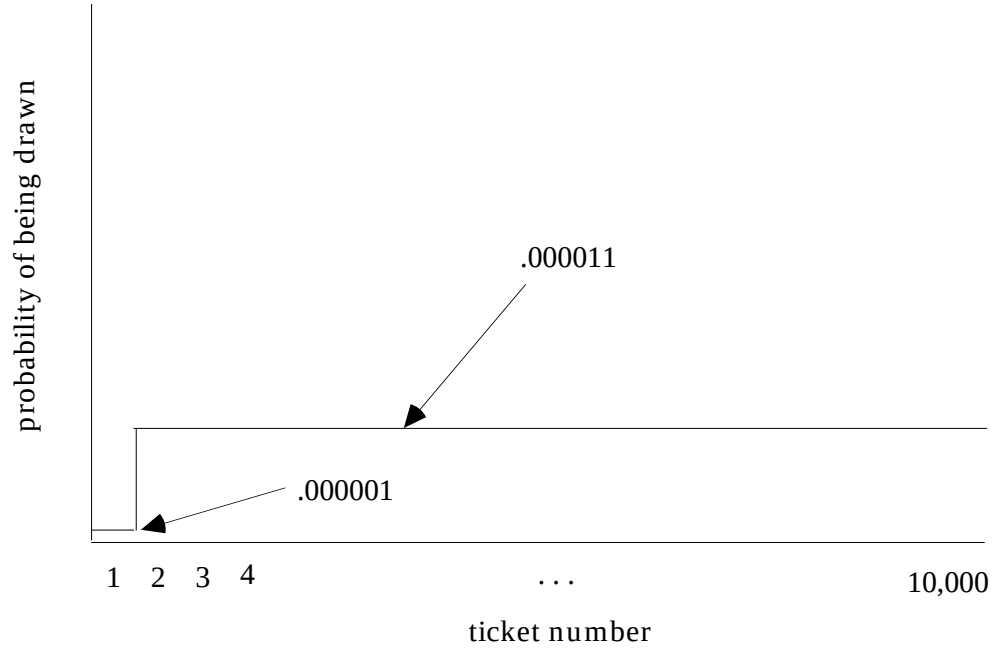


Figure 3. Lottery 3.

Now contrast lottery 2 with lottery 3, which consists of 10,000 tickets. In lottery 3, the probability of ticket 1 being drawn is still .000001, but the probability of any other ticket being drawn is .000011. This is diagramed as in figure 3. It may still be reasonable to infer that ticket 1 will not be drawn, but, and this is the crucial observation, the justification for this conclusion does not

seem to be nearly so strong. This is because although we have the same defeasible argument for "$\sim T_1$", the reasons involved in the counterargument for "T_1" are now much better, being based upon a probability of .999989. They are still not strong enough to defeat the argument for "$\sim T_1$" outright, but they seem to weaken the justification. Thus the degree of justification for "$\sim T_1$" is lower in lottery 3 than it is in lottery 2. The difference between lottery 2 and lottery 3 seems to illustrate that defeaters that are too weak to defeat a conclusion outright may still lower the degree of justification. In other words, they act as *diminishers*.

In my (1990) I argued that reasoning similar to this treatment of biased lotteries is what underlies statistical induction. In statistical induction, having observed a sample of *A*'s and found that the proportion of members of the sample that are *B*'s is *r*, we infer defeasibly that the probability of an arbitrary *A* being a *B* is approximately *r*, i.e., lies in an interval $[r-\delta,r+\delta]$ around the observed relative frequency *r*. The logic of the reasoning that allows us to conclude this is the same as that involved in reasoning about biased lotteries. We know that the actual probability *p* is in the interval [0,1]. This is like knowing that some ticket will be drawn. If the sample is large, then for each choice of *p* the probability of getting the observed relative frequency is low given that *p* is the actual probability, but for some choices of *p* (those further from *r*) it is lower than for others. So we can reason as in the biased lottery that the actual probability does not lie on the tails of the bell curve.

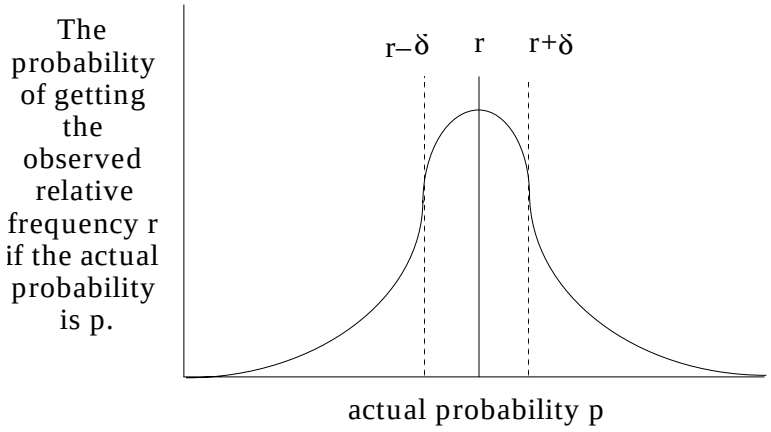


Figure 4. Statistical Induction

The argument for diminishers is based on intuitive examples. In particular, I have appealed to biased lotteries. If one ticket is *much* more improbable than the others, it seems reasonable to conclude that it will not be drawn. But if it is only minutely less probable, that does not seem reasonable. This seems equally true for arguments in general. If the argument for a defeater is only minutely weaker than the argument for a conclusion, it does not seem reasonable to regard the conclusion as unscathed. These observations seem to me to be unassailable.

7. The Mathematics of Diminishers

Given an undefeated argument for *P* and an otherwise undefeated weaker argument for ~*P*,

the degree of justification for *P* should be the argument strength of the first argument decremented by an amount determined by the argument strength of the second argument. That is, there should be a function ♦ such that given two arguments that rebut one another, if their strengths are *x* and *y*, the degree of justification for the conclusion of the former is x ♦ *y*. Let us see what we can determine about the mathematical properties of ♦.

I will assume that degrees of justification can be measured using real numbers, possibly augmented with ∞, i.e., the "extended real numbers". More precisely, I assume that degrees of justification fall in some interval [o,θ] where $0 \leq o \leq \theta \leq \infty$. o corresponds to no justification, and θ to perfect justification, presumably only possible for necessary truths. For now we can leave it unspecified whether o = 0 and whether θ is finite. I will make the following assumptions:

(A1) ♦ is continuous on the interval [o,θ].

(A2) If $\theta > \alpha > \beta > o$ then $\alpha > \alpha \blacklozenge \beta > o$. (Diminishers diminish, without completely defeating.)

(A3) If $\theta > \alpha > \beta > \gamma > o$ then $\alpha \blacklozenge \beta < \alpha \blacklozenge \gamma$ and $\alpha \blacklozenge \gamma > \beta \blacklozenge \gamma$. (♦ is monotonic)

(A4) If If $\theta \geq \alpha \geq \beta \geq o$ then $\beta \blacklozenge \alpha = o$.

(A5) If $\theta \geq \alpha > o$ then $\alpha \blacklozenge o = \alpha$.

(A6) If $\theta > \alpha$ and β and γ are in [o,θ] then $(\alpha \blacklozenge \beta) \blacklozenge \gamma = (\alpha \blacklozenge \gamma) \blacklozenge \beta$. (The order in which diminishers are applied to an argument makes no difference to the resulting degree of justification.)

I take it that these assumptions are relatively uncontroversial. Note that they are purely algebraic assumptions. In general, the arithmetical properties of diminishers will depend both on the algebraic properties of diminishers and on the particular way in which we assign numbers to quantities of justifications. The latter is a *scale*. Different scales can yield different arithmetical properties, but the differences won't be substantive because they just reflect different ways of assigning numbers to items in a domain having the same fixed algebraic structure. For example, we can measure temperatures in Fahrenheit or Celsius, and we get different values, but these differences do not reflect different claims about the world. It is significant then that (A1) – (A6) are independent of any scale. The principal result of this section will be to show that these purely algebraic assumptions entail the existence of a scale for which ♦ has a particularly simple arithmetical representation.

Theorem 1: If $\theta > \gamma \geq \alpha$ and $\gamma \blacklozenge \alpha \leq \beta$ then $\gamma \blacklozenge \beta \leq \alpha$.
Proof: Suppose $\gamma \blacklozenge \alpha \leq \beta$. Then by (A4), $(\gamma \blacklozenge \alpha) \blacklozenge \beta = o$, so by (A6), $(\gamma \blacklozenge \beta) \blacklozenge \alpha = o$, so by (A2), $\gamma \blacklozenge \beta \leq \alpha$. ■

Theorem 2: If $\theta > \gamma \geq \alpha$ and $\gamma \blacklozenge \alpha = \beta$ then $\gamma \blacklozenge \beta = \alpha$.
Proof: Suppose $\gamma \blacklozenge \alpha = \beta$. Then by theorem 1, $\gamma \blacklozenge \beta \leq \alpha$. Suppose $\gamma \blacklozenge \beta < \alpha$. By (A5), $\gamma \blacklozenge o \geq \alpha$, so by continuity (A1), there is a λ such that $\lambda \leq \beta$ and $\gamma \blacklozenge \lambda = \alpha$. Then by theorem 1, $\gamma \blacklozenge \alpha \leq \lambda < \beta$,

which contradicts the supposition that $\gamma \blacklozenge \alpha = \beta$. ■

Theorem 3: If $\theta > \alpha \geq \beta \geq o$ then $\alpha \blacklozenge (\alpha \blacklozenge \beta) = \beta$. (Diminishing α by the difference between α and β leaves degree of justification β.)
Proof: Suppose $(\alpha \blacklozenge \beta) = (\alpha \blacklozenge \beta)$, so by theorem 2, $\alpha \blacklozenge (\alpha \blacklozenge \beta) = \beta$. ■

Theorem 4: If $\theta \geq \alpha \geq \beta \geq \gamma \geq o$ then $(\alpha \blacklozenge \gamma) \blacklozenge (\alpha \blacklozenge \beta) = \beta \blacklozenge \gamma$.
Proof: $\beta \blacklozenge \gamma = (\alpha \blacklozenge (\alpha \blacklozenge \beta)) \blacklozenge \gamma$ (by theorem 3)
$= (\alpha \blacklozenge \gamma) \blacklozenge (\alpha \blacklozenge \beta)$ (by (A6)) ■

Theorem 5: If $\theta \geq \alpha \geq \beta \geq \gamma \geq o$ then $(\alpha \blacklozenge \gamma) \blacklozenge (\beta \blacklozenge \gamma) = \alpha \blacklozenge \beta$.
Proof: By theorem 4, $(\alpha \blacklozenge \gamma) \blacklozenge (\alpha \blacklozenge \beta) = \beta \blacklozenge \gamma$, so by theorem 2, $(\alpha \blacklozenge \gamma) \blacklozenge (\beta \blacklozenge \gamma) = \alpha \blacklozenge \beta$. ■

We can define an inverse for $\blacklozenge$. Given (A7), the following is well-defined:

Definition: If $\theta > \alpha$ and $\theta > \beta$ then

$$\alpha \oplus \beta = \begin{cases} \text{glb}\{x \mid x \blacklozenge \beta \geq \alpha\} & \text{if } (\exists x)\, x \blacklozenge \beta \geq \alpha. \\ \theta & \text{otherwise.} \end{cases}$$

Theorem 6: If $\theta > \alpha \oplus \beta$ then $(\alpha \oplus \beta) \blacklozenge \beta = \alpha$.
Proof: By (A4), $\beta \blacklozenge \beta = o$. If $\theta > \alpha \oplus \beta$ then there is a γ such that $\gamma \blacklozenge \alpha \geq \beta$. By (A1), $\blacklozenge$ is continuous, so there is a γ such that $\gamma \blacklozenge \alpha = \beta$. By definition, $\alpha \oplus \beta = \gamma$. ■

Theorem 7: If $\theta > \alpha \geq \beta \geq o$ then $(\alpha \blacklozenge \beta) \oplus \beta = \alpha$.
Proof: $\alpha \blacklozenge \beta \leq \alpha \blacklozenge \beta$, so $(\alpha \blacklozenge \beta) \oplus \beta = \text{glb}\{x \mid x \blacklozenge \beta \geq \alpha \blacklozenge \beta\}$. By (A3), this is α. ■

Theorem 8: If $\theta > \alpha \geq \gamma > o$ and $\alpha \blacklozenge \gamma = \beta$ then $\beta \oplus \gamma = \alpha$.
Proof: Suppose $\alpha \blacklozenge \gamma = \beta$. By theorem 7, $\beta \oplus \gamma = (\alpha \blacklozenge \gamma) \oplus \gamma = \alpha$. ■

Theorem 9: If $\theta > \alpha \geq \gamma > o$ and $\theta > \beta \oplus \gamma$, if $\beta \oplus \gamma = \alpha$ then $\alpha \blacklozenge \gamma = \beta$.
Proof: Suppose $\beta \oplus \gamma = \alpha$. By theorem 6, $\alpha \blacklozenge \gamma = (\beta \oplus \gamma) \blacklozenge \gamma = \beta$. ■

Theorem 10: If $\theta > \alpha$ and $\theta > \beta$ then $\alpha \oplus \beta = \beta \oplus \alpha$.
Proof: Suppose $\alpha \oplus \beta < \theta$. By (A3), $\alpha \geq \alpha \blacklozenge \beta$, so by theorem 8, $\alpha \oplus \beta \geq \alpha$. By (A4), $\alpha \geq \beta \blacklozenge \beta = o$, so by theorem 8, $\alpha \oplus \beta \geq \beta$. Thus by theorem 9, if $\alpha \oplus \beta = \gamma$ then $\gamma \blacklozenge \beta = \alpha$, so by theorem 2, $\gamma \blacklozenge \alpha = \beta$, and hence by theorem 8, $\beta \oplus \alpha = \gamma$. So $\alpha \oplus \beta = \beta \oplus \alpha$.

Suppose $\alpha \oplus \beta = \theta$. If $\beta \oplus \alpha < \theta$, then as in the previous paragraph, $\alpha \oplus \beta = \beta \oplus \alpha \neq \theta$. So $\beta \oplus \alpha = \theta$. ■

Theorem 11: If $\theta > \alpha \geq \beta$ and $\theta > (\alpha \oplus \gamma)$ then $(\alpha \blacklozenge \beta) \oplus \gamma = (\alpha \oplus \gamma) \blacklozenge \beta$.
Proof: Suppose $(\alpha \oplus \gamma) < \theta$. Then $(\alpha \blacklozenge \beta) \oplus \gamma < \theta$. So by theorems 8 and 9, $(\alpha \blacklozenge \beta) \oplus \gamma = x$ iff $x \blacklozenge \gamma = \alpha \blacklozenge \beta$.

$((\alpha \oplus \gamma) \blacklozenge \beta) \blacklozenge \gamma = ((\alpha \oplus \gamma) \blacklozenge \gamma) \blacklozenge \beta$ (by A6)
$= \alpha \blacklozenge \beta$ (by theorem 6).

So $(\alpha \blacklozenge \beta) \oplus \gamma = (\alpha \oplus \gamma) \blacklozenge \beta$. ■

Theorem 12: If $\theta > (\alpha \oplus \beta)$ then $(\alpha \oplus \beta) \blacklozenge \alpha = \beta$.
Proof: By theorem 10, $\alpha \oplus \beta = \beta \oplus \alpha$. $\alpha \oplus \beta \geq \alpha$, so by theorems 8 and 9, $(\alpha \oplus \beta) \blacklozenge \alpha = \beta$ iff $\alpha \oplus \beta = \beta \oplus \alpha$, and the latter holds by theorem 10. ■

Theorem 13: If $\theta \geq \alpha \geq \beta \geq o$ then $(\exists \varepsilon)\ \alpha = \beta \oplus \varepsilon$.
Proof: If $\alpha = \beta$, then by (A5) and theorem 8, $\alpha = \beta \oplus o$. If $\alpha > \beta$ then by theorem 3, $\beta = \alpha \blacklozenge (\alpha \blacklozenge \beta)$, so by theorem 8, $\alpha = \beta \oplus (\alpha \blacklozenge \beta)$. ■

Theorem 14: If $\theta > (\alpha \oplus \gamma)$ and $\theta > (\beta \oplus \gamma)$ then $(\alpha \oplus \gamma) \blacklozenge (\beta \oplus \gamma) = \alpha \blacklozenge \beta$.
Proof: By theorem 6, $\alpha = (\alpha \oplus \gamma) \blacklozenge \gamma$ and $\beta = (\beta \oplus \gamma) \blacklozenge \gamma$. So

$\alpha \blacklozenge \beta = ((\alpha \oplus \gamma) \blacklozenge \gamma) \blacklozenge ((\beta \oplus \gamma) \blacklozenge \gamma) = (\alpha \oplus \gamma) \blacklozenge (\beta \oplus \gamma)$ by theorem 5. ■

Theorem 15: If $\theta > (\alpha \oplus \beta)$ and $\theta > (\beta \oplus \gamma)$ then $(\alpha \oplus \beta) \oplus \gamma = \alpha \oplus (\beta \oplus \gamma)$.
Proof: $\alpha \blacklozenge \gamma = (\alpha \oplus \beta) \blacklozenge (\gamma \oplus \beta)$ (by theorem 14)
$= (\alpha \oplus \beta) \blacklozenge (\beta \oplus \gamma)$ (by theorem 10).
So $\alpha \oplus \beta = (\alpha \blacklozenge \gamma) \oplus (\beta \oplus \gamma)$ (by theorem 8)
$= (\alpha \oplus (\beta \oplus \gamma)) \blacklozenge \gamma$ (by theorem 11).
So $(\alpha \oplus (\beta \oplus \gamma)) = (\alpha \oplus \beta) \oplus \gamma$ (by theorem 8). ■

Next let us define sequential diminishing. Informally,

$\alpha \blacklozenge_n \beta = (\ldots (\alpha \blacklozenge \beta) \blacklozenge \beta) \ldots \blacklozenge \beta)$ (for n ♦'s)

The formal definition is recursive:

Definition:

$\alpha \blacklozenge_0 \beta = \alpha$.

$\alpha \blacklozenge_{n-1} \beta = (\alpha \blacklozenge_n \beta) \blacklozenge \beta$.

We can also define multiplication by an integer. Informally,

$\alpha \odot n = (\ldots (\alpha \oplus \alpha) \oplus \ldots \oplus \alpha)$ (for n α's).

The formal definition is again recursive:

Definition:

$\alpha \odot 0 = o$.

$\alpha \odot (n+1) = (\alpha \odot n) \oplus \alpha$.

There is an obvious connection between $\blacklozenge_n$ and $\odot$:

Theorem 16: If $\theta > \alpha$, $\theta > \beta$ and $(\alpha \blacklozenge_n \beta) > o$ then $\alpha = (\alpha \blacklozenge_n \beta) \oplus (\beta \odot n)$.
Proof by induction on *n*, using theorem 7. ■

Theorem 17: If $\theta > \alpha$, $\theta > \beta$ and $(\alpha \blacklozenge_n \beta) > o$ then $(\alpha \blacklozenge_n \beta) = \alpha \blacklozenge (\beta \odot n)$.
Proof: By theorems 8 and 16. ■

We can also define division by an integer, for $n > 0$:

Definition:

$(\alpha \oslash n) = \text{glb}\{x \mid x \odot n \geq \alpha\}$ (provided $n > 0$)

Theorem 18: If $\theta > \alpha$ and $n > 0$ then $((\alpha \oslash n) \odot n) = \alpha$.

Proof: $\oplus$ is continuous, so $(x \odot n)$ is as well relative to *x*. By (A4), $o \blacklozenge o = o$, so by theorem 8, $o \oplus o = o$. Thus $(o \odot n) \leq \alpha$. And $(\alpha \odot \alpha) \geq \alpha$, so $(\alpha \odot n) \geq \alpha$. Thus there is a β such that $(\beta \odot n) = \alpha$, and hence $(\alpha \oslash n) = \beta$. ■

Theorem 19: If $\theta > \alpha$ and $n > 0$ and $\alpha > \beta$ then $(\alpha \odot n) > (\beta \odot n)$.
Proof by induction on *n* using (A3). ■

Theorem 20: If $\theta > \alpha$ and $n > 0$ then $(\alpha \oslash n) = \beta$ iff $(\beta \odot n) = \alpha$.

Proof: If $(\alpha \oslash n) = \beta$ then by theorem 15, $(\beta \odot n) = \alpha$. Conversely, by theorem 19, if $\beta \neq (\alpha \oslash n)$ then $(\beta \odot n) \neq ((\alpha \oslash n) \odot n) = \alpha$. ■

Theorem 21: If $\theta > \alpha$ and $n > 0$ then $(\alpha \oslash n) \oplus (\beta \oslash n) = ((\alpha \oplus \beta) \oslash n)$.

Proof: By theorem 20, $((\alpha \oplus \beta) \oslash n) = (\alpha \oslash n) \oplus (\beta \oslash n)$ iff $((\alpha \oslash n) \oplus (\beta \oslash n)) \odot n = \alpha \oplus \beta$.

$((\alpha \oslash n) \oplus (\beta \oslash n)) \odot n$
$= ((\alpha \oslash n) \oplus (\beta \oslash n)) \oplus \ldots \oplus ((\alpha \oslash n) \oplus (\beta \oslash n))$
$= ((\alpha \oslash n) \oplus \ldots \oplus (\alpha \oslash n)) \oplus ((\beta \oslash n) \oplus \ldots \oplus (\beta \oslash n))$ (by theorems 10 and 15)
$= ((\alpha \oslash n) \odot n) \oplus ((\beta \oslash n) \odot n)$
$= \alpha \oplus \beta$. (by theorem 16). ■

Theorem 22: If $\theta > \alpha$ then $\alpha \odot mn = (\alpha \odot m) \odot n$.
Proof by induction on *n*. Informally,

$\alpha \odot mn = \alpha \oplus \ldots (mn \text{ times}) \ldots \oplus \alpha$
$= (\alpha \oplus \ldots (m \text{ times}) \ldots \oplus \alpha) \oplus \ldots (n \text{ times}) \ldots \oplus (\alpha \oplus \ldots (m \text{ times}) \ldots \oplus \alpha)$
$= (\alpha \odot m) \oplus \ldots (n \text{ times}) \ldots \oplus (\alpha \odot m) = (\alpha \odot m) \odot n$. ■

Theorem 23: If $\theta > \alpha$ and $m > 0$ then $(\alpha \div m) \odot mn = \alpha \odot n$.

Proof: By theorem 22, $(\alpha \div m) \odot mn = ((\alpha \div m) \odot m) \odot n = \alpha \odot n$ (by theorem 18). ■

Theorem 24: If $\theta > \alpha$, $m > 0$ and $n > 0$ then $(\alpha \div m) = ((\alpha \odot n) \div mn)$.

Proof: By theorem 23, $\alpha \odot n = (\alpha \div m) \odot mn$, so by theorem 20, $(\alpha \div m) = ((\alpha \odot n) \div mn)$. ■

Theorem 25: If $\theta > \alpha$, $n > 0$ and $l > 0$ then $(\alpha \odot m) \div n = (\alpha \odot ml) \div nl$.

Proof: By theorem 24, $(\alpha \odot m) \div n = ((\alpha \odot m) \odot l) \div nl = (\alpha \odot ml) \div nl$ (theorem 22). ■

Theorem 26: If $\theta > \alpha$, $n > 0$ and $l > 0$ then $((\alpha \odot m) \div n) \oplus ((\alpha \odot k) \div l) = (\alpha \odot (ml + kn)) \div nl$.

Proof: By theorem 25, $((\alpha \odot m) \div n) \oplus ((\alpha \odot k) \div l) = ((\alpha \odot ml) \div nl) \oplus ((\alpha \odot kn) \div nl) = (\alpha \odot (ml + kn)) \div nl$ (theorem 21). ■

Theorem 27: If $\theta > \alpha > o$ then $((\alpha \odot m) \div n) = ((\alpha \odot k) \div l)$ iff $m / n = k / l$.

Proof: Suppose $((\alpha \odot m) \div n) = ((\alpha \odot k) \div l)$. By theorem 21, $((\alpha \odot m) \div n) = ((\alpha \odot ml) \div nl)$ and $((\alpha \odot k) \div l) = ((\alpha \odot kn) \div nl)$. Hence $((\alpha \odot ml) \div nl) = ((\alpha \odot kn) \div nl)$. Then it follows by (A3) that $(\alpha \odot ml) = (\alpha \odot kn)$. Then by (A3) again, $ml = kn$, so $m / n = k / l$. ■

It follows from the preceding theorems that no degrees of justification less than θ are "infinitely larger than" others, in the following sense:

Theorem 28: If $\theta > \alpha$ and $\theta > \beta > o$ then then there is an n such that $\beta \odot n \geq \alpha$.

Proof: Suppose otherwise. Let $\beta = \text{lub}\{y \mid (\forall n \in \omega)\ (y \odot n) < \alpha\}$. So if $\varepsilon > 0$, $(\forall n \in \omega)\ ((\beta - \varepsilon) \odot n) < \alpha$. $\beta \odot n = \lim_{\varepsilon \to 0} ((\beta - \varepsilon) \odot n) \leq \alpha$. This holds for every $n \in \omega$, so $(\beta \odot (n+1)) \leq \alpha$. But $(\beta \odot n) < (\beta \odot (n+1)) \leq \alpha$, so $(\forall n \in \omega)\ (\beta \odot n) < \alpha$. However, by theorem 22, $((\beta \oplus \beta) \odot n) = (\beta \odot 2n) < \alpha$, and $\beta \oplus \beta > \beta$. As β is the least upper bound, $(\exists n \in \omega)\ ((\beta \oplus \beta) \odot n) \geq \alpha$. So we have a contradiction. ■

Theorem 29: If $\theta > \alpha$ and $\theta > \beta > o$ then then there is an n such that $(\alpha \div n) \leq \beta$.
Proof: Theorems 28 and 20. ■

Given an arbitrarily chosen "unit" ι such that $o < \iota < \theta$, we can define "quasi-rationals" and "quasi-irrationals":

α is *quasi-rational* (relative to ι) iff there are $m,n \in \omega$ such that $\alpha = ((\iota \odot m) \div n)$.

α is *quasi-irrational* (relative to ι) iff α is not quasi-rational.

We can prove that quasi-irrational numbers can be systematically approximated by sequences of

quasi-rationals:

Theorem 30: If $\theta > \alpha$ and α is quasi-irrational then there is a sequence $\{\alpha_i\}_{i\in\omega}$ of quasi-rationals such that (1) for every $n\in\omega$, $\alpha_n < \alpha$, and (2) for every $\delta > o$ there is an $n\in\omega$ such that $\alpha \blacklozenge \alpha_n \leq \delta$.

Proof: It follows from theorem 29 that for every $\delta > o$, there is an $n\in\omega$ such that $\delta \geq (\iota \oslash n)$. Let $\delta_n = (\iota \oslash n)$. So for every $\delta > o$ there is an n such that $\delta_n \leq \delta$.

By theorem 28, there is an m such that $(\delta_n \odot m) \geq \alpha$. Let m_n be the first such m. Let $\alpha_n = (\delta_n \odot (m_n - 1))$. Then $\alpha_n < \alpha$ and $(\alpha_n \oplus \delta_n) \geq \alpha$. So $\alpha \blacklozenge \alpha_n \leq \delta_n$. So for every $\delta > o$, there is an $n\in\omega$ such that $\alpha \blacklozenge \alpha_n \leq \delta$. ■

In other words, $\alpha = \lim_{n\to\infty} \alpha_n$. Note that we actually proved the stronger theorem:

Theorem 31: If $\theta > \alpha$, $\iota > o$, and α is quasi-irrational relative to ι, then there is a sequence $\{m_n\}_{n\in\omega}$ of integers such that $\alpha = \lim_{n\to\infty} ((\iota \odot m_n) \oslash n)$.

Theorem 32: If α and β are quasi-irrational, $\alpha = \lim_{n\to\infty} ((\iota \odot m_n) \oslash n)$, and $\beta = \lim_{n\to\infty} ((\iota \odot k_n) \oslash n)$, then $(\alpha \oplus \beta) = \lim_{n\to\infty} ((\iota \odot (m_n + k_n)) \oslash n)$.

Proof: By theorem 15, $(\iota \odot m_n) \oplus (\iota \odot k_n) = (\iota \odot (m_n + k_n))$. ■

Now we can prove our principal representation theorem. Assumptions (A1) – (A6) are completely algebraic, and as such are independent of the scale we use for assigning numbers to degrees of justification. Once we settle upon a scale, it then becomes possible to investigate the arithmetical properties of $\blacklozenge$, $\oplus$, etc. on that scale. Our representation theorem tells us that there is a scale on which degrees of justification occupy the interval $[0,\infty]$ and $\oplus$ is ordinary addition. A *transform* is simply a one-one mapping from one scale onto another. Let us define:

Definition:

g is an *additive transform* iff there is an extended real number r ($r \leq \infty$) such that

(1) g maps $[o,\theta]$ 1–1 onto $[0,r]$;

(2) $g(o) = 0$;

(3) $g(\theta) = r$;

(4) If $\theta > (\alpha \oplus \beta)$ then $g(\alpha \oplus \beta) = g(\alpha) + g(\beta)$.

Our representation theorem is then:

Theorem 33: If (A1) – (A6) hold then there exists an additive transform.

Proof: Choose ι such that $\theta > \iota > o$. Set $g(o) = 0$.

If α is quasi-rational relative to ι and $\alpha = ((\iota \odot m) \oslash n)$, set $g(\alpha) = m/n$. This is unique by theorem 27.

If α is quasi-irrational relative to ι and $\{m_n\}_{n\in\omega}$ is a sequence such that for each $n\in\omega$, $((\iota \odot m_n) \oslash n) < \alpha$, and $\alpha = \lim_{n\to\infty} ((\iota \odot m_n) \oslash n)$, set $g(\alpha) = \lim_{n\to\infty} (m_n / n)$. Again, this is unique by

theorem 27.

It follows from theorem 26 and 32 that if $\theta > (\alpha \oplus \beta)$ then $g(\alpha \oplus \beta) = g(\alpha) + g(\beta)$.

If for some $\alpha, \beta < \theta$, $(\alpha \oplus \beta) = \theta$, set $g(\theta) = \text{lub}\{g(\alpha) + g(\beta) \mid \alpha, \beta < \theta \text{ and } (\alpha \oplus \beta) = \theta\}$. Otherwise set $g(\theta) = \infty$. ■

For additive transforms, ♦ also has a very simple representation. Let us define:

Definition:

$$x \sim y = \begin{cases} x - y \text{ if } y < x < \infty \\ 0 \text{ otherwise} \end{cases}$$

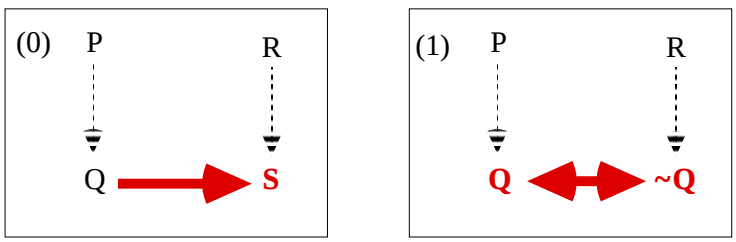
Theorem 34: If (A1) – (A6) hold, g is an additive transform, and $\theta > \alpha$, then $g(\alpha \blacklozenge \beta) = g(\alpha) \sim g(\beta)$.

Proof: If $\theta > \alpha > \beta$, $\alpha \blacklozenge \beta = \gamma$ iff $\gamma \oplus \beta = \alpha$. Thus $g(\alpha) = g((\alpha \blacklozenge \beta) \oplus \beta) = g(\alpha \blacklozenge \beta) + g(\beta)$, and hence $g(\alpha \blacklozenge \beta) = g(\alpha) - g(\beta)$. If $\alpha \leq \beta$ then by (A4), $\alpha \blacklozenge \beta = o$, so $g(\alpha \blacklozenge \beta) = 0$. ■

Let us say that a scale for degrees of justification is *additive* iff degrees of justification occupy some interval $[0,r]$, $\oplus$ is represented by +, and ♦ is represented by ~. It follows from theorems 33 and 34 that there exists an additive scale for degrees of justification. I will henceforth assume that we are dealing with such a scale. A substantial question remains how to actually assign numbers to degrees of justification so as to define an additive scale. This will be taken up in section ten. The scale produced there has the result that $\theta = \infty$.

8. Computing Defeat Statuses

Now let us return to the question of how diminishers should affect the defeat status computation. I assume that rather than merely assigning "defeated" or "undefeated", a status assignment should now assign numerical degrees of justification $j(P)$ to nodes. For this purpose, I will also assume that we are employing an additive scale for measuring degrees of justification. How is $j(P)$ determined? In the following examples, unless I explicitly say otherwise, I assume that the reason-strengths are at least as great as the degrees of justification of the initial nodes so that they can be ignored in computing degrees of justification. If we consider a very simple inference-graph like (0), it it follows from theorem 34 that we should have $j(Q) = j(P)$, $j(S) = j(R) \sim j(Q)$. So if $j(P) \geq j(R)$, S is defeated, otherwise it is diminished.



Consider the marginally more complicated inference-graph (1). Here there seem to be two

possibilities regarding how the degrees of justification are to be computed:

(a) We could have $j(Q) = j(P) \sim j(\sim Q)$ and $j(\sim Q) = j(R) \sim j(Q)$.
(b) We could have $j(Q) = j(P) \sim j(R)$ and $j(\sim Q) = j(R) \sim j(P)$.

These seem to be the only two possibilities for this simple inference-graph. However, (a) is not a genuine possibility. We have the following theorem:

Theorem 35: If $j(P) > j(R)$, $j(Q) = j(P) \sim j(\sim Q)$ and $j(\sim Q) = j(R) \sim j(Q)$ then $j(Q) = j(P)$ and $j(\sim Q) = 0$.

Proof: Suppose $j(Q) \neq j(P)$. As $j(Q) = j(P) \sim j(\sim Q)$, $j(Q) < j(P)$, so $j(\sim Q) \neq 0$. Then as $j(\sim Q) = j(R) \sim j(Q)$, $j(\sim Q) = j(R) - j(Q) \leq R$. By assumption, $j(P) > j(R)$, so $j(P) > j(\sim Q)$, and hence $j(Q) = j(P) \sim j(\sim Q) = j(P) - j(\sim Q) = j(P) - j(R) + j(Q)$. Thus $j(R) = j(P)$, which is impossible given the assumption that $j(P) > j(R)$. So by reductio, $j(Q) = j(P)$. As $j(P) > j(R)$, $j(P) > 0$. But $j(Q) = j(P) \sim j(\sim Q)$, so $j(\sim Q) = 0$. ■

Thus (a) is incompatible with diminishing. That leaves only (b) as a possible computation of defeat statuses in inference-graph (1). That is,

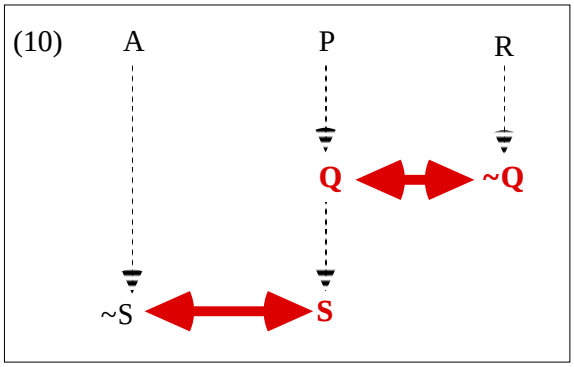
$j(Q) = j(P) \sim j(R)$ and $j(\sim Q) = j(R) \sim j(P)$.

This means that in computing $j(Q)$, we take the strength of the argument supporting it, in this case determined by $j(P)$, and then subtract the strength of the argument supporting the defeater, i.e., $j(R)$. We do not subtract the strength of the defeater itself, i.e., we do not subtract $j(\sim Q)$.

If we apply (b) to the case in which $j(P) = j(R)$, we get a single status assignment in which $j(Q) = j(\sim Q) = 0$. This has a surprising consequence. The semantics for defeasible reasoning described in section one, as well as default logic, the stable model semantics, circumscription, and almost all standard semantics for defeasible reasoning and nonmonotonic logic, support what I have called (1987) "presumptive defeat".[12] For example, consider inference-graph (10). A defeated conclusion like Q that is assigned "defeated" in some status assignment and "undefeated" in another retains the ability to defeat. In the case of inference-graph (10) this has the consequence that S is assigned "defeated" in those status-assignments in which Q is assigned "defeated", but S is assigned "undefeated" and $\sim S$ is assigned "defeated" in those status-assignments in which Q is assigned "undefeated". Touretzky, Horty, and Thomason (1987) called this "ambiguity propagation", and Makinson and Schlechta (1991) called such arguments "Zombie arguments" (they are dead, but they can still get you). However, computation (b) precludes presumptive defeat. It entails that there is a single status assignment in which $j(S) = j(Q) = j(\sim Q) = 0$, and $j(\sim S) = j(A)$. So Q, $\sim Q$, and S are all defeated, and $\sim S$ is undefeated. Is this the right answer? Consider an example: Jones and Smith hear one weather forecast, but disagree about whether rain was predicted (Q). Q gives one reason to believe it will rain (S). You didn't hear the first forecast, but

[12] The only semantics I know about that does not support presumptive defeat are certain versions of Nute's (1992) defeasible logic. See also Covington, Nute, and Vellino (1997), and Nute (1999).

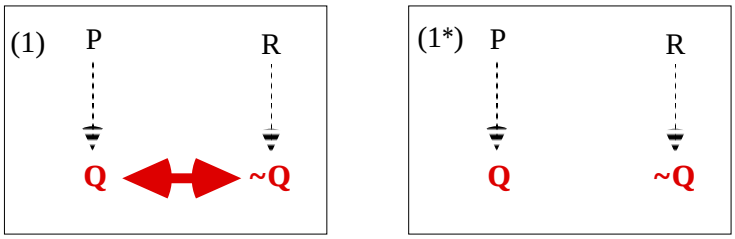
you hear another forecast (*A*), which says it will not rain. Should you believe ~*S*? I have some inclination to think you should, but I don't find the answer obvious, nor have I ever found another example of presumptive defeat where the answer is obvious. At a recent workshop on defeasible reasoning held at the University of Georgia, I found there to be no consensus among the participants as to the intuitive status of presumptive defeat.



In the absence of clear intuitions, how can we decide whether presumptive defeat is a genuine phenomenon of defeasible reasoning, or an undesirable artifact of existing semantics? The preceding considerations constitute a rather strong argument for the conclusion that presumptive defeat is incompatible with diminishing. If it is granted that a correct semantics must accommodate diminishing, this means in turn that most standard semantics for defeasible reasoning produce incorrect assessments of inference-graph (10) even when all premises have the same degree of justification and all reason-strengths are the same. It is generally assumed that by ignoring variations in degrees of justification and reason-strengths we are simplifying the problem of constructing a semantics for defeasible reasons, but the present considerations indicate that by doing so we may also obscure phenomena that have important implications for the semantics even in this simplified case.

The fact that we get a single assignment of defeat statuses in inference-graph (1) suggests that we are not really computing status assignments. Rather, we are computing degrees of justification directly. The appeal to status assignments was motivated by the thought that we could not compute degrees of justification recursively because a node *P* of an inference-graph can have *P*-dependent defeaters (i.e., defeaters that are inference / defeat-descendants of *P*.) But what the present approach may yield is a way of doing a different kind of recursive computation of degrees of justification. In inference-graph (1), we cannot compute $j(Q)$ without having a value for ~*Q*, and we cannot compute $j(\sim Q)$ without having a value for *Q*. So we cannot do a straightforward recursive computation of degrees of justification. However, the values required for *Q* and ~*Q* need not be their degrees of justification. Indeed, they cannot be because those would be zero for each. In computing $j(Q)$, what we want to subtract from $j(P)$ is not $j(\sim Q)$ but rather a measure of the strength of the argument for ~*Q*. The value of the latter should be $j(R)$. This measure can be obtained by removing the mutual defeat between the two arguments, as in inference-graph (1*), and computing $j(Q)$ and $j(\sim Q)$ in the new inference-graph. Call those values $j^*(Q)$ and $j^*(\sim Q)$. Then returning to inference-graph (1), the values we subtract when we compute $j(Q)$ and $j(\sim Q)$ are $j^*(\sim Q)$ and $j^*(Q)$. That is, $j(Q) = j^*(Q) \sim j^*(\sim Q) = j(P) \sim j(R)$, and $j(\sim Q) = j^*(\sim Q) \sim j^*(Q) = j(R) \sim j(P)$. In constructing inference-graph (1*), we remove two defeat-links. Each link

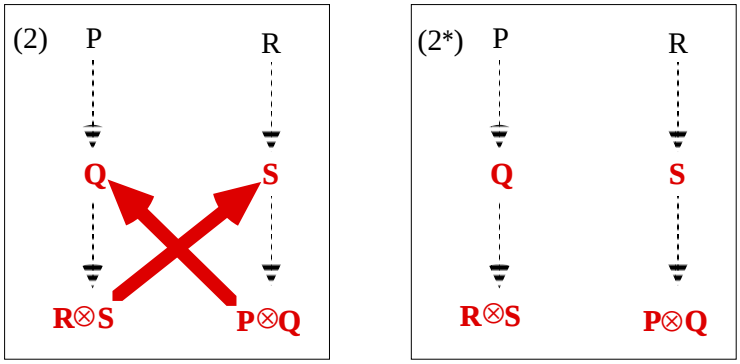
has the characteristic that removing it results in *Q* no longer having a *Q*-dependent defeater, and also in ~*Q* no longer having a ~*Q*-dependent defeater.



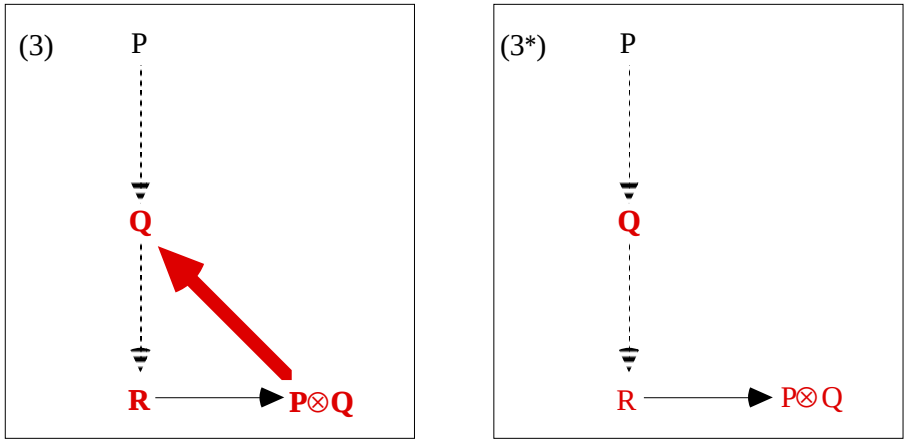
Now consider inference-graph (2). Again, there seem to be two ways the computation of degrees of justification might go. Presumably $j(R \otimes S) = j(Q)$ and $j(P \otimes Q) = j(S)$. Then we might have either:

(a) $j(S) = j(R) \sim j(R \otimes S) = j(R) \sim j(Q)$ and $j(Q) = j(P) \sim j(P \otimes Q) = j(P) \sim j(S)$; or
(b) $j(S) = j(R) \sim j(P)$ and $j(Q) = j(P) \sim j(R)$.

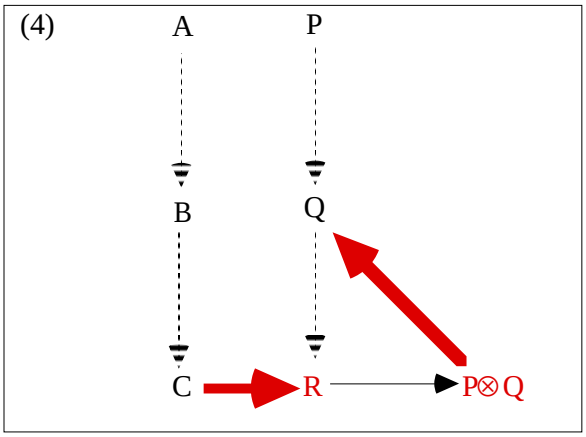
As in theorem 35, (a) is incompatible with diminishing, so (b) seems to be the only possibility. This means that in computing *j(S)* we begin with the strength of the argument supporting *S*, i.e., *j(R)*, and then subtract the strength of the argument supporting the defeater *R*⊗*S* would have in the absence of the defeater *P*⊗*Q* that is obtained from *S*. We compute *j(Q)* analogously. This is the same thing as computing *j(R*⊗*S)* and *j(P*⊗*S)* in inference-graph (2*) (call the resulting values *j**(*R*⊗*S*) and *j**(*P*⊗*Q*)) and then setting *j(Q)* = *j**(*Q*) ~ *j**(*P*⊗*Q*) and *j(S)* = *j**(*S*) ~ *j**(*R*⊗*S*). Again, the defeat-links we remove in constructing inference-graph (2*) are those such that if we remove either, *Q* no longer has a *Q*-dependent defeater, and similarly for *S*.



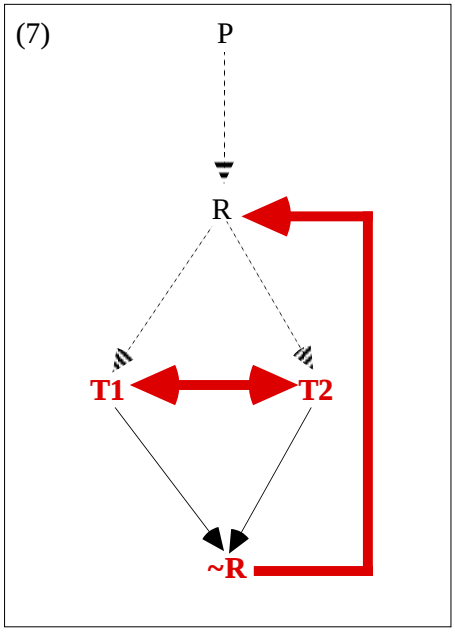
Consider inference-graph (3). Presumably $j(P \otimes Q) = j(R) = j(Q) = 0$. We can get that by ruling that $j(Q) = j(P) \sim j(P)$. So here we begin with the strength of the argument supporting *Q*, i.e., *j(P)*, and then subtract the strength the argument supporting *P*⊗*Q* would have in the absence of the defeater (itself) that is obtained from *Q*. This is the value assigned to (*P*⊗*Q*) in inference-graph (3*). Again, the defeat-link we remove in constructing (3*) is the only defeat-link whose removal results in *Q* no longer having a *Q*-dependent defeater.

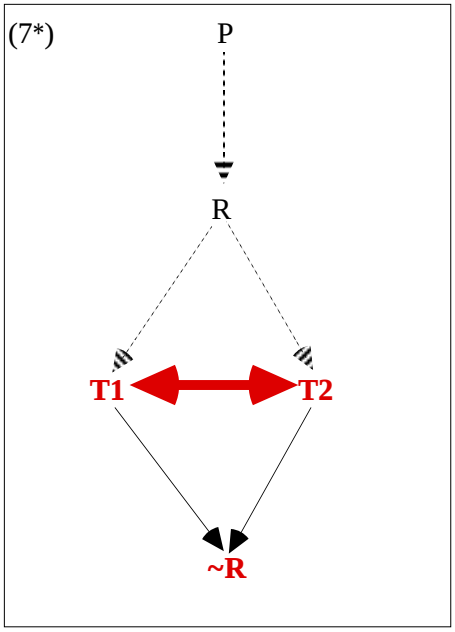


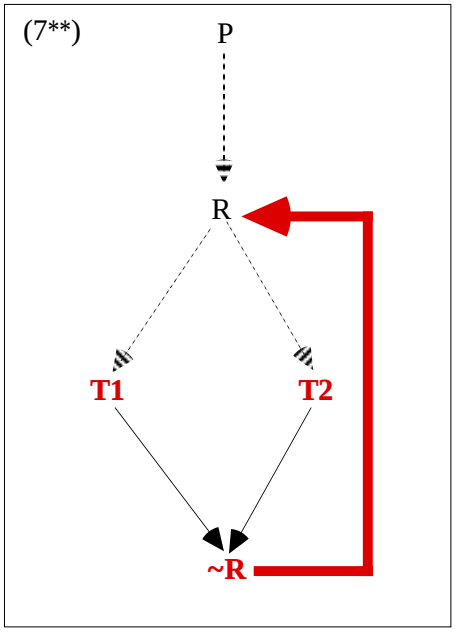
Inference-graph (4) is analogous. In (4) we compute $j^*(P\otimes Q)$ by removing the defeat-link between $P\otimes Q$ and Q. So $j^*(P\otimes Q) = j(P) \sim j(A)$. That has the result that if $j(A) = j(P)$ then $j^*(P\otimes Q) = j^*(R) = 0$ and so $j(Q) = j^*(Q) \sim j^*(P\otimes Q) = j(P)$. I will continue to refer to $j^*(P\otimes Q)$ as the argument-strength of $(P\otimes Q)$, but note that this is a somewhat more complex notion than the argument-strength discussed in section four. In the present sense, argument-strengths take account not only of the strengths of the premises and inference-schemes but also of the effect of those defeaters that are independent of the node being evaluated.



Finally, consider the lottery paradox paradox, in the guise of inference-graph (7). We construct (7*) by removing the only defeat-link whose removal results in R no longer having an R-dependent defeater. In (7*), the triangle consisting of R, $T1$ and $T2$ is analogous to inference-graph (1), and so $j^*(T1) = j^*(R) \sim j^*(R) = 0$, and $j^*(T2) = j^*(R) \sim j^*(R) = 0$. $j^*(\sim R) = \min\{j^*(T1), j^*(T2)\} = 0$. Then $j(R) = j^*(R) \sim j^*(\sim R) = j(P)$.

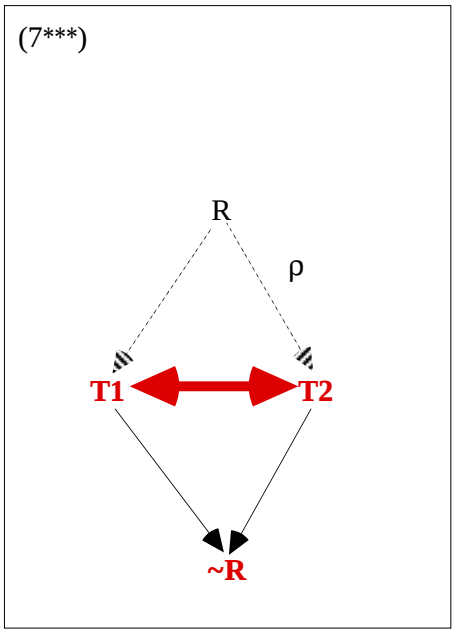


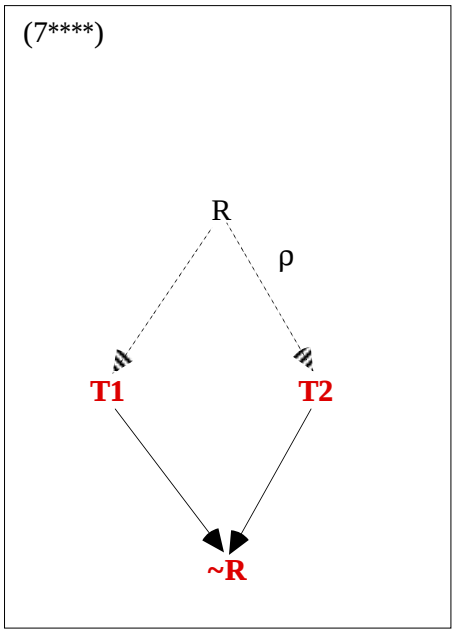




When we turn to the computation of *j(T1)* and *j(T2)* in (7), we encounter an additional complication. Removing the defeat-links whose removal results in *T1* no longer having a *T1*-dependent defeater produces inference-graph (7**). On the strength of the preceding examples we might expect that *j(T1)* = *j***(*T1*) ~ *j***(*T2*). However, inference-graph (7**) is analogous to inference-graph (3). Accordingly, *j***(*T1*) = *j****(*T2*) = *j***(*R*) = *j***(~*R*) = 0. This produces the intuitively correct answer that *j(T1)* = 0, but it seems to do so in the wrong way. To see this, let us modify the example slightly by taking it to represent a biased lottery. More precisely, let us suppose that the reason-strength of the reason supporting the inference of *T1* from *R* is at least as great as the degree of justification of *P* (and hence of *R*), but suppose the reason-strength ρ of the reason supporting the inference of *T2* from *R* is less than the degree of justification of *P*. In

this case *T2* should be defeated, and hence ~*R* should be defeated, but *T1* should only be diminished. However, this change does not affect the computation of degrees of justification in inference-graph (7**). We still get that *j***(*T1*) = *j***(*T2*) = *j*(*R*) = *j***(~*R*) = 0.





The source of the difficulty is that the computation of degrees of justification is not being done recursively. We should first compute the degree of justification of *R*, as above. Then holding that degree of justification fixed we should go on to compute the degrees of justification of its inference-children, in this case *T1* and *T2*. This amounts to regarding *R* as an initial node whose degree of justification is that computed in inference-graph (7). This yields inference-graph (7***). The degrees of justification for *T1* and *T2* are then those computed in inference-graph (7***). That computation proceeds by constructing inference-graph (7****) and computing *j*****(*T1*) = *j*(*R*), *j*****(*T2*) = ρ, and so *j*(*T1*) = *j****(*T1*) = *j*****(*T1*) ~ *j*****(*T2*) = *j*(*R*) ~ ρ. Analogously, *j*(*T2*) = *j*****(*T2*) ~ *j*****(*T1*) = 0. Continuing recursively, *j*(~*R*) = min{*j*(*T1*),*j*(*T2*)} = 0.

The defeat status computation that emerges from these examples proceeds in accordance with two rules. We begin with an inference-graph *G*. Where φ is a node of *G*, let *j*(φ,*G*) be the degree of justification of φ in *G*. The first rule governs the case in which *P* has no *P*-dependent defeaters. In that case, the computation proceeds in accordance with the originally proposed rules (1)–(3). This has the effect of computing degrees of justification in terms of the degrees of justification of the basis of *P* and the defeaters of *P*. The second rule governs the case in which *P* has *P*-dependent defeaters. In that case the computation of the degree of justification of *P* proceeds instead in terms of the argument-strength for *P* and maximal argument-strength of the *P*-dependent defeaters. Those argument-strengths are computed by constructing a new inference-graph G_P by removing each defeat-link of *G* whose removal results in *P* no longer having a *P*-dependent defeater. More generally, there can be parallel routes from one node to another, with the result that we must delete multiple defeat-links to ensure that *P* no longer has a *P*-dependent defeater. So let us define:

Definition: A defeat-link is *P-critical* in an inference-graph G iff it is a member of some minimal set of defeat-links such that the removal of all the links in the set results in P no longer having a *P*-dependent defeater.

Definition: G_P is the inference-graph that results from removing all *P*-critical defeat-links from G and making all members of the node-basis of *P* and all *P*-independent nodes φ initial with $j(\varphi,G_P) = j(\varphi,G)$.

The argument strengths in G are then the degrees of justification in G_P.

Putting this altogether, the two rules are as follows (where $\max(\varnothing) = 0$) :

(DJ1) If *P* is inferred from the basis $\{B_1,...,B_n\}$ in an inference-graph G in accordance with a reason-scheme of strength ρ, $D_1,...,D_k$ are the defeaters for *P*, and no D_i is *P*-dependent, then

$$j(P,G) = \min\{\rho, j(B_1,G),...,j(B_n,G)\} \sim \max\{j(D_1,G),...,j(D_k,G)\}.$$

(DJ2) If *P* has *P*-dependent defeaters $D_1,...,D_k$ in Gthen

$$j(P,G) = j(P,G_P) \sim \max\{j(D_1,G_P),...,j(D_k,G_P)\}.$$

The general idea is that the computation of degrees of justification is made recursive by appealing to argument strengths rather than degrees of justification in ungrounded cases. Argument strengths are computed by computing degrees of justification in simpler inference-graphs from which the source of ungroundedness has been removed. The result is a recursive computation of degrees of justification.

9. Collaborative Defeat

P-dependent defeaters for *P* are those lying on an *inference/defeat-path* from *P* to *P*, where the latter is a path consisting of support-links and defeat-links and terminating with a defeat-link. One minimal way of cutting all such paths is to remove all the final defeat-links, so *P*-dependent defeaters for *P* are automatically *P*-critical. (There may be other *P*-critical defeaters as well, e.g., one lying on every inference / defeat path from *P* to *P*.) *P*-critical defeaters are *P*-dependent, so:

Theorem 40: A defeater for *P* is *P*-critical iff it is *P*-dependent.

It follows that in constructing G_P, the defeaters for *P* that are removed are precisely the *P*-dependent defeaters for *P*. Thus the rules (DJ1) and (DJ2) have the effect of dividing the defeaters of *P* into two sets — those that are *P*-dependent and those that are not. These two sets of defeaters then affect $j(P,G)$ separately. Those that are *P*-independent will diminish the value of *P* in G_P, reducing $j(P,G_P)$, and then those that are *P*-dependent will further diminish the value of *P* in G. So if D_1 is the most strongly supported defeater that is *P*-independent, and D_2 is the most strongly supported defeater that is *P*-dependent, the result will be that $j(P,G) = j(P,G_P) \sim j(D_2,G_P)$ $= (\min\{\rho, j(B_1,G_P),...,j(B_n,G_P)\} \sim j(D_1,G_P)) \sim j(D_2,G_P)) = (\min\{\rho, j(B_1,G),...,j(B_n,G)\} \sim j(D_1,G)) \sim j(D_2,G_P))$

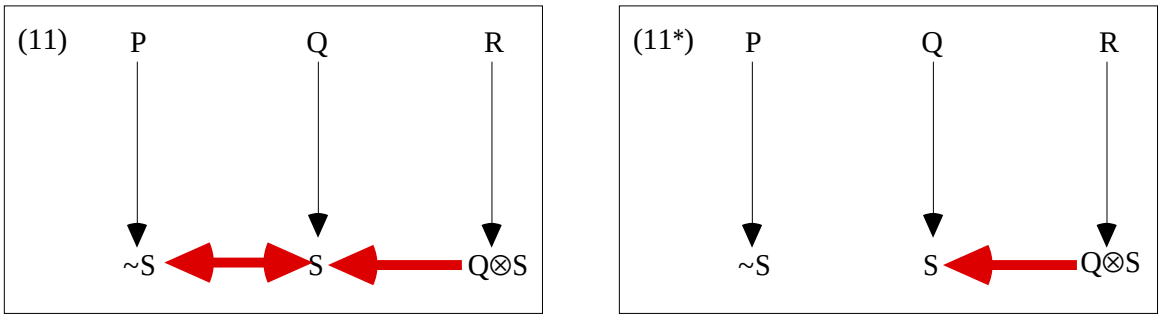
$= \min\{\rho, j(B_1,G),...,j(B_n,G)\} \sim (j(D_1,G) + j(D_2,G))$. In this way, we can replace the sequential application of (DJ2) and (DJ1) by the application of a single principle:

(DJ) If P is inferred from the basis $\{B_1,...,B_n\}$ in an inference-graph G in accordance with a reason-scheme of strength ρ, $D_1,...,D_k$ are the P-independent defeaters for P, and $D_{k+1},...,D_m$ are the P-dependent defeaters of P, then
$j(P,G) = \min\{\rho, j(B_1,G),...,j(B_n,G)\} \sim [\max\{j(D_1,G),...,j(D_k,G)\} + \max\{j(D_{k+1},G_P),...,j(D_m,G_P)\}]$.

This produces a kind of double counting of defeaters. It has the consequence that although no single defeater may be sufficient to defeat the inference to P, two defeaters can accomplish that by acting in unison. I will call this *collaborative defeat*.[13]

Collaborative defeat might seem suspect. However, I will now argue that there are examples which illustrate that it actually occurs in defeasible reasoning. As an autobiographical note, I first became convinced of the need for collaborative defeat on the basis of examples like those I will present below. This was long before I had a theory of diminishers that entailed collaborative defeat. My expectation was that I would have to construct an initial theory to accommodate diminishers, and then embellish it in some rather ad hoc way to make it compatible with collaborative defeat. Thus I am delighted to find that collaborative defeat simply falls out of my theory of diminishers without any ad hoc treatment.

To set the stage, note that you cannot have two rebutting defeaters such that one is P-dependent and the other is not. Rebutting defeaters of P are always P-dependent. This is because rebutting defeat is symmetrical, so if you can follow defeat-links in one direction, you can follow them back again. Thus you can have one defeater that is P-dependent and another that is not only if one of them is an undercutting defeater. The general form is that of inference-graph (11). To compute $j(S,G_{11})$, we construct inference-graph (11*). $j(S,G_{11*}) = j(Q,G_{11}) \sim j(R,G_{11})$, and so $j(S,G_{11}) = (j(Q,G_{11}) \sim j(R,G_{11})) \sim j(P,G_{11}) = j(Q,G_{11}) \sim (j(P,G_{11}) + j(R,G_{11}))$.



To confirm that the computation in inference-graph (11) is correct, we must consider examples that mix rebutting defeat and undercutting defeat. One case in which this occurs is causal reasoning. Consider the Yale Shooting Problem. This problem has played an important role in

[13] Normally, the collaborating defeaters will be reasons for different conclusions, viz., $\sim P$ and $(P \otimes Q)$. But in some cases they can both be reasons for $(P \otimes Q)$, in which case this has a similar flavor to the accrual of reasons. It is not anything that simple, however, because each defeater is the strongest defeater (not a sum) from a whole class of defeaters — the dependent and independent ones.

discussions of the Frame Problem.[14] In the Yale Shooting Problem, we are given that a gun is initially loaded. Then it is pointed at Jones and the trigger is pulled. We suppose we know (simplistically) that if a loaded gun is pointed at someone and the trigger pulled, that person will shortly become dead. The conclusion we are supposed to draw in this case is that Jones will die. The Yale Shooting Problem is the problem of showing how this conclusion can be justified. There are two difficulties. First, our reason for thinking that the gun is loaded when the trigger is pulled is that it was loaded moments earlier when we checked it. So we are employing temporal projection. Temporal projection provides a defeasible reason for thinking that if something is true at one time then it will still be true later. Some form of temporal projection has been endorsed by most recent authors discussing the frame problem.[15] Using temporal projection, we can construct:

Argument 1 — for the conclusion that Jones will be dead after the shooting, based upon causal knowledge and the temporal projection that the gun will still be loaded when the trigger is pulled.

However, as Hanks and McDermott (1986, 1987) were the first to note, with the help of temporal projection we can also construct a second argument supporting a conflicting conclusion:

Argument 2 — for the conclusion that Jones will still be alive after the shooting, based upon temporal projection from the fact that Jones was initially alive.

In the absence of further arguments, these arguments will defeat each other, leaving us with no justified conclusion to draw about the state of Jones' health. To get the intuitively correct answer, we need a third argument:

Argument 3 — supporting an undercutting defeater for argument 1.

The Yale Shooting Problem is resolved by explaining the details of argument 3. I have proposed such a solution in my (1998). For present purposes, the details are not important. Suffice it to say that the undercutter gives inferences based upon causal knowledge priority over those based upon temporal projection and turns on the premise that the gun is still loaded when the trigger is pulled.

A problem derives from the fact that the strength of an inference by temporal projection decreases as the time interval increases. That is, temporal projection gives us a reason for thinking that things won't change, but the greater the time interval the weaker the reason. This is the *temporal decay* of temporal projection discussed at the beginning of section three (see my 1998). If we ignore the temporal decay of temporal projection, the foregoing constitutes a solution to the Yale Shooting Problem. But now suppose, as will typically be the case, that we observe Jones to be alive at a later time (normally, right up to the time we pull the trigger) than we observe the

---

[14] Hanks and McDermott (1986).

[15] This was proposed by Sandewall (1972), and subsequently endorsed by McDermott (1982), McCarthy (1986), and virtually all subsequent authors.

gun to be loaded. I cannot observe the latter at the time I pull the trigger without shooting myself in the face. In this case the strengths of arguments 1 and 3, depending as they do on inferring that the gun is still loaded when the trigger is pulled, may both be less than the strength of argument 2.

The temporal profile we should get is the following. If we observe the gun to be loaded long before observing Jones to be alive (e.g., years ago), our justification for believing Jones to remain alive might be somewhat weakened but not defeated. On the other hand, if we observe the gun to be loaded just shortly before observing Jones to be alive, we should be able to conclude that Jones will die. The intuitive rationale for this profile seems to be as follows. First, we have the undefeated argument 3 for the undercutting defeater for argument 2, but it is weaker than argument 2. Instead of defeating argument 2 outright, it weakens it seriously. This leaves us with only a weak reason for thinking that Jones remains alive. Then argument 1 provides a reason for thinking Jones is dead. If argument 1 turns upon a temporal projection from the distant past, it will not be strong enough to defeat even the weakened argument 2, but if the temporal projection is from a recent observation then argument 1 will be strong enough to defeat the weakened argument 2. This is exactly the computation that results from collaborative defeat.

The upshot is that what seemed initially like a suspicious consequence of (DJ1) and (DJ2) turns out to be a very important logical phenomenon that is often crucial for computing defeat statuses correctly.

10. Measuring Strengths

I am assuming that reason-strengths and degrees of justification are being measured using using an additive scale, as discussed in section seven. On an additive scale, degrees of justification are measured using the extended reals, and ♦ is represented by $\sim$. However, nothing has been said about how this additive scale is defined. If we are to take strength seriously, we must have some way of measuring it. One way is to compare reasons with a set of standard equally good reasons that have numerical values associated with them in some determinant way. I propose to do that by taking the set of standard reasons to consist of instances of the statistical syllogism (SS). For any proposition *p*, we can construct a standardized argument for $\sim p$ on the basis of the pair of suppositions "$\text{prob}(F/G) \geq r$ & Gc" and "$(p \leftrightarrow \sim Fc)$":

1. Suppose $\text{prob}(F/G) \geq r$ & Gc.
2. Suppose $(p \leftrightarrow \sim Fc)$.
3. Fc from 1.
4. $\sim p$ from 2,3.

where the strength of the argument is a function of *r*. We can measure the strength of a defeasible reason for *p* in terms of that value of *r* such that the conflicting argument from the suppositions "$\text{prob}(F/G) \geq r$ & Gc" and "$(p \leftrightarrow \sim Fc)$" exactly counteracts it. The value *r* determines the reason-strength in the sense that the reason-strength is some function $\boldsymbol{dj}(r)$ of *r*. Any monotonic increasing function $\boldsymbol{dj}(r)$ $(0.5 \leq r \leq 1.0)$ will provide a scale. For example, we could define $\boldsymbol{dj}(r) = r$, or $\boldsymbol{dj}(r) = r^2$, or $\boldsymbol{dj}(r) = (1 - e^{-r})$, etc. However, we are looking specifically for an additive scale. By identifying

♦ with **~**, we have required reason-strength to be a cardinal measure that can be meaningfully added and subtracted. Adding and subtracting reason-strengths may not be the same thing as adding and subtracting the corresponding probabilities. To illustrate the general point, should it turn out (it won't) that the reason-strength corresponding to a probability r is given by $\log(r)$, then adding reason-strengths would be equivalent to multiplying probabilities rather than adding them.

I do not have an a priori argument to offer regarding what function $\boldsymbol{dj}(r)$ produces an additive scale. The only way to determine this is to look for proposals that work plausibly in concrete examples. Perhaps the most illuminating example is that of the biased lotteries diagrammed in figures 2 and 3. This example quickly rules out the simplest proposal, which is that $\boldsymbol{dj}(r) = r$. Suppose reason-strengths could be identified with the corresponding probabilities. In lottery 2, the probability of ticket 1 being drawn is .000001, and the probability of any other ticket being drawn is .111111. We wanted to conclude in this case that we are justified in believing that ticket 1 will not be drawn. The probability corresponding to the argument-strength for this conclusion is .999999, and the probability corresponding to the counter-argument for the conclusion that ticket 1 will be drawn (because no other ticket will) is .888889.

This example makes it obvious that the scale defined by setting $\boldsymbol{dj}(r) = r$ is not an additive scale. The difference between the probabilities associated with the argument and counter-argument in this example is .11111, which is a very low probability. If probabilities and degrees of justification could be identified, i.e., $\boldsymbol{dj}(r) = r$, subtraction would produce too low a degree of justification for it to be reasonable to believe that ticket 1 will not be drawn. So apparently we cannot compute degrees of justification by adding and subtracting the probabilities themselves.

There is statistical lore suggesting that in probabilistic reasoning degrees of justification can be compared in terms of likelihood ratios.[16] When (as in the biased lotteries) we have an argument for P based on a probability r, and a weaker argument for $\sim P$ based on a probability r^*, the likelihood ratio is $(1 - r)/(1 - r^*)$. The suggestion is that the degree of justification for P is determined by the likelihood ratio. For example, in lottery 2 the likelihood ratio is .000009, while in lottery 3 it is .09. Note that likelihood ratios are defined so that higher likelihood ratios correspond to lower degrees of justification. An equivalent but more intuitive way of measuring degrees of justification is by using the inverse of the likelihood ratios.

The reason likelihood ratios produce plausible comparisons in the case of the biased lotteries is that by looking at $1/(1 - r)$, when the probabilities are close to 1 the differences in likelihood produced by small differences in probability are large. For example, $1/(1 - .999999)$ is 1,000,000, but $1/(1 - .999989) = 90{,}909$ and $1/(1 - .888889) = 9$.

In my (1990) I suggested that likelihood ratios yield the intuitively correct answers in many cases of statistical and inductive reasoning. If we define $\boldsymbol{dj}(r) = \log(1/(1 - r))$, then the result of subtracting degrees of justification is the same as taking the logarithm of the inverse of the likelihood ratio, i.e., $\boldsymbol{dj}(r) - \boldsymbol{dj}(r^*) = \log(1/(1 - r)) - \log(1/(1 - r^*)) = \log((1 - r^*)/(1 - r))$.

However, this definition of $\boldsymbol{dj}$ fails to satisfy the obvious constraint that for $r \leq .5$, $\boldsymbol{dj}(r) = 0$. Furthermore, for $r > .5$, $\boldsymbol{dj}(r)$ should approach 0 as r approaches .5. In my (2001) I proposed to

---

[16] This is known as the likelihood principle. It is due to R. A. Fisher (1922), and versions of it have been endorsed by a variety of authors, including G. A. Barnard (1949 and 1966), Alan Birnbaum (1962), A. W. F. Edwards (1972), and Ian Hacking (1965).

handle this by defining $\boldsymbol{dj}(r) = \log(.5) - \log(1 - r)$ (for $r \geq .5$). But I have subsequently come to realize that this way of normalizing degrees of justification does not work. Degrees of justification are supposed to correspond to probabilities in the sense that if d is any possible degree of justification then there is an r between .5 and 1 such that $d = \boldsymbol{dj}(r)$. Furthermore, if d and d^* are possible degrees of justification and $d \geq d^*$ then $d - d^*$ should also be a possible degree of justification. Thus if $x \geq y$ and x and y are between .5 and 1, then there should be an r between .5 and 1 such that $\boldsymbol{dj}(x) - \boldsymbol{dj}(y) = \boldsymbol{dj}(r)$. If we consider the lottery 2, this would require that

$$\boldsymbol{dj}(r) = \log(.5) - \log(1 - r) = \boldsymbol{dj}(.999999) - \boldsymbol{dj}(.999989) = -1.0414$$

and hence $r = -4.5$, which is not an allowable value.

I propose instead that we normalize $\boldsymbol{dj}(r)$ by defining:

$$\boldsymbol{dj}(r) = \begin{cases} \log((1 / (1 - r)) - 1) \text{ for } r \geq .5 \\ 0 \text{ for } r < .5 \end{cases}$$

Equivalently, $\boldsymbol{dj}(r) = \log(r / (1 - r))$ for $r \geq .5$. For $.5 \leq r \leq 1$, this is a monotonic increasing function, and its range is $[0,\infty]$. Furthermore, it has the consequence that when x and y are between .5 and 1 and $x \geq y$ then there is an r between .5 and 1 such that $\boldsymbol{dj}(r) = \boldsymbol{dj}(x) - \boldsymbol{dj}(y)$:

Theorem 37: When x and y are between .5 and 1 and $x \geq y$ then there is an r between .5 and 1 such that $\boldsymbol{dj}(r) = \boldsymbol{dj}(x) - \boldsymbol{dj}(y)$.

Proof:

$$\boldsymbol{dj}(x) - \boldsymbol{dj}(y) = \log(x / (1 - x)) - \log(y / (1 - y)) = \log(\frac{x \cdot (1 - y)}{y \cdot (1 - x)}).$$

When x and y are between .5 and 1 and $x \geq y$, the function

$$\frac{x \cdot (1 - y)}{y \cdot (1 - x)}$$

is a monotonic increasing function of x and a monotonic decreasing function of y. Its value lies in the interval $[1,\infty]$. Thus the values of $\boldsymbol{dj}(x) - \boldsymbol{dj}(y)$ lie in the interval $[0,\infty]$. $0 = \boldsymbol{dj}(.5)$ and $\infty = \boldsymbol{dj}(1)$, and if $0 < x < \infty$ then there is a y between .5 and 1 such that $x = \boldsymbol{dj}(y)$. ■

When r and r^* are close to 1, $\boldsymbol{dj}(r) - \boldsymbol{dj}(r^*)$ is approximately equal to $\log((1 - r^*) / (1 - r))$, so this definition has the effect of evaluating comparative reason-strengths in terms of likelihood ratios. On the other hand, if r and r^* are closer to .5 then the comparison can diverge from likelihood ratios. However, this seems to be required to make the normalization work, and the values of $\boldsymbol{dj}(r)$ produced by this definition are intuitively reasonable. For r and r^* close to 1, if $(1 - s) = (1 - r^*) / (1 - r)$ then $\boldsymbol{dj}(r) - \boldsymbol{dj}(r^*)$ is approximately equal to $\boldsymbol{dj}(s)$. In other words, the diminished justification is approximately equal to the justification deriving from an undiminished instance of the statistical syllogism based on probability s. For example, in lottery 2, $(1 - r^*) / (1 - r) = 1 / 900{,}000$ which is $1 - .9999989$. Precisely,

$$\boldsymbol{dj}(.999999) - \boldsymbol{dj}(.888889) = 5.0969 = \boldsymbol{dj}(.999992).$$

So the justification, after diminishing, for the conclusion that ticket 1 will not be drawn is the

same as that deriving from an instance of the statistical syllogism based on the very high probability .999992. This accords with our intuition that we are justified in concluding that ticket 1 will not be drawn in lottery 2. It is analogous to concluding (defeasibly) that an event having a 1 in 900,000 chance of occurring will not occur. When we turn to lottery 3, $(1 - r^*)/(1 - r) = 1/11$ which is 1 – .90909. Precisely,

$$\boldsymbol{dj}(.999999) - \boldsymbol{dj}(.999989) = 1.0414 = \boldsymbol{dj}(.9167).$$

Here the diminished justification is the same as that deriving from an instance of the statistical syllogism based on the probability .9167. This accords with our intuition that we are no longer justified in concluding that ticket 1 will not be drawn — if something has a 1 in 11 chance of occurring, we cannot reasonably conclude (even defeasibly) that it will definitely not occur.

An important consequence of this definition of $\boldsymbol{dj}(r)$ is that computations of the values of linear combinations of degrees of justification can be done entirely "inside" the logarithms. For example, $\boldsymbol{dj}(x) + \boldsymbol{dj}(y) - \boldsymbol{dj}(z) = \log([x/(1-x)]\cdot[y/(1-y)]\cdot[(1-z)/z])$. Thus comparisons of the values of linear combinations are independent of the base of the logarithm.

My proposal for calibrating degrees of justification is thus that we employ a scale defined as follows:

Definition: ***The SS scale*****:**

If X is a defeasible reason for p, the strength of this reason is $\log(r/(1-r))$ where r is that real number such that an argument for $\sim p$ based upon the suppositions "$\text{prob}(F/G) \geq r$ & Gc" and "$(p \leftrightarrow \sim Fc)$" and employing the statistical syllogism exactly counteracts the argument for p based upon the supposition X.

Note that $\boldsymbol{dj}(1) = \infty$. So on the SS scale, the strongest reasons have infinite reason-strength. This could create problems if we ever wanted to subtract the strengths of such arguments from each other, because $\infty - \infty$ is undefined, but in fact we will never have occasion to do that.

The SS scale only tells us how to assign reason-strengths to reasons. The degrees of justification for conclusions are determined jointly by this and the degrees of justification of the initial nodes in arguments. The latter will be determined by the source of the initial nodes. For example, in a human-like agent, the initial nodes will all be based upon perception. If the source of an initial node has probability p associated with it, it is natural to extend the SS scale by requiring that the degree of justification of the node is $\log(p/(1-p))$. The value of p may be a default assumption of the cognitive architecture.

No proof has been given that the SS scale is additive. Indeed, it is hard to see what could count as a proof. All we can do is give evidence for that claim by showing that computing degrees of justification in terms of it by identifying ♦ with ~ yields intuitively reasonable results.

It is worth noting that the assumption that the SS scale is additive has the following interesting consequence:

Theorem 38: If the SS scale is an additive scale, then if $\theta > \alpha$ and $\theta > \beta$, there is a γ such that $\theta > \gamma$ and $\gamma \blacklozenge \alpha = \beta$.

Proof: The consequent of the conditional is a purely algebraic consequence that is independent

of scale, so if it holds for the SS scale when ♦ is identified with ~ then it holds for any scale. And it holds for the SS scale because $\theta = \infty$. Let $\infty > \alpha,\beta \geq 0$. Then there are real numbers r,s in the interval [0,1) such that $\alpha = \boldsymbol{dj}(r) = \log(r/(1-r))$ and $\beta = \boldsymbol{dj}(r^*) = \log(r^*/(1-r^*))$. Suppose $\gamma = \boldsymbol{dj}(s) = \log(s/(1-s))$. Then $\gamma \blacklozenge \alpha = \beta$ iff $\log(s/(1-s)) - \log(r/(1-r)) = \log(r^*/(1-r^*))$, i.e., iff

$$\frac{s(1-r)}{r(1-s)} = \frac{r^*}{(1-r^*)}$$

This holds iff $s = \dfrac{rr^*}{rr^* + (1-r)(1-r^*)}$

If $r,r^* < 1$ then $(1-r)(1-r^*) > 0$, so $0 < s < 1$. Thus there is always such an s, and hence such a γ. ■

Note that in the preceding proof, $\boldsymbol{dj}(s) = \boldsymbol{dj}(r) + \boldsymbol{dj}(r^*)$. So we also have the following (equivalent) theorem:

Theorem 39: If the SS scale is an additive scale, then if $\theta > \alpha$ and $\theta > \beta$, $(\alpha \oplus \beta) < \theta$.

What theorems 38 and 39 tell us is that for any beliefs P and Q, no matter how highly justified there are, it is possible to have a proposition R such that (1) there is an argument for $\sim R$ that is a strong as the justification of Q, but (2) there is an argument for R that is so strongly justified that even after diminishing, the justification of R is as great as that of P. It is not immediately obvious that this is true, but if the SS scale is indeed additive then this can be illustrated with biased lotteries. If the degree of justification of P is $\boldsymbol{dj}(r^*)$ and the degree of justification of Q is $\boldsymbol{dj}(r)$, just imagine a biased lottery in which the probability of ticket 1 being drawn is $(1-s)$, i.e., $1 - (rr^*/(rr^* + (1-r)(1-r^*)))$ and the probability of any other ticket being drawn is $(1-r)$.

11. Simplifying the Computation

Principles (DJ1) and (DJ2) provide a recursive characterization of degrees of justification relative to an inference-graph. However, this characterization does not lend itself well to implementation because it requires the construction of modified inference-graphs, which would be computationally expensive. The objective of this section is to produce an equivalent recursive characterization that appeals only to the given inference-graph.

11.1 Bypasses

Recall that a defeat-link or support-link extends from its *root* to its *target*. The root of a defeat-link is a single node, and the root of a support-link is a set of nodes. Let us define precisely:

Definition: An *inference/defeat-path* from a node φ to a node θ is a sequence of support-links and defeat-links such that (1) φ is the root of the first link in the path; (2) θ is the last link in the path; (3) the root of each link after the first member of the path is the target of the preceding link; (4) the path does not contain an internal loop, i.e., no two links in the path have the same target.

Definition: θ is φ-*dependent* iff there is an inference / defeat-path from φ to θ.

We defined a defeat-link to be φ–critical in an inference-graph *G* iff it is a member of some minimal set of defeat-links such that the removal of all the links in the set results in φ no longer having a φ–dependent defeater in *G*. We are specifically interested in inference / defeat-paths from a node to its own defeat-links, so let us define:

Definition: A *circular inference/defeat-path* from a node φ to itself is an inference / defeat-path from φ to a defeat-link for φ.

Then a defeat-link is φ-critical iff it is a member of a minimal set such that removing all the defeat-links in the set suffices to cut all the circular inference / defeat-paths from φ to φ. A necessary condition for a defeat-link *L* to be φ–critical is that it lie on such a circular path. In general, there can be diverging and reconverging paths with several "parallel" defeat-links, as in figure 6. In figure 6, removing the defeat-link D_1 suffices to cut both circular paths. But the set $\{D_1,D_2\}$ of parallel defeat-links is also a minimal set of defeat-links such that the removal of all the links in the set suffices to cut all the circular inference / defeat-paths from φ to φ. Thus in figure 6, all of the defeat-links are φ–critical. However, lying on a circular inference / defeat-path is not a sufficient condition for being φ-critical. A defeat-link on a circular inference / defeat-path from φ to φ can fail to be φ–critical is when there is a path around it consisting entirely of support-links, as diagrammed in figure 7. In this case, you must remove D_1 to cut both paths, and once you have done that, removing D_2 is a gratuitous additional deletion. So D_2 is not contained in a minimal set of deletions sufficient for cutting all the circular inference / defeat-paths from φ to φ, and hence D_2 is not φ-critical. This phenomenon is also illustrated by inference-graph (7), and it is crucial to the computation of degrees of justification in that inference-graph that such defeat-links not be regarded as φ-critical. It turns out that this is the only way a defeat-link on a circular inference / defeat-path can fail to be φ-critical, as will now be proven.

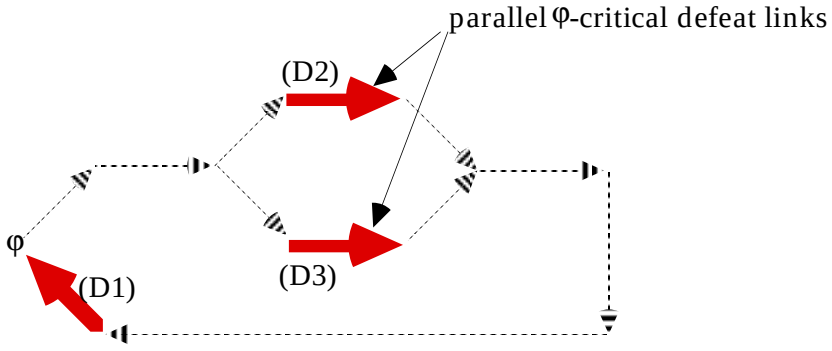


Figure 6. Parallel φ-critical defeat-links

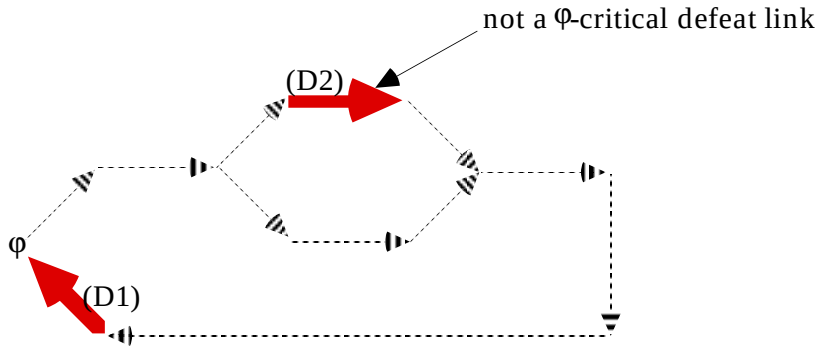


Figure 7. Defeat link that is not φ-critical

Let us say that a node α *precedes* a node β on an inference / defeat-path iff α and β both lie on the path and either α = β or the path contains a subpath originating on α and terminating on β. *Node-ancestors* of a node are nodes that can be reached by following support-links backwards. It will be convenient to define:

Definition: A defeat-link *L can be bypassed* on an inference / defeat-path μ in *G* iff there is a node α preceding the root of *L* on μ and a node β preceded by the target of *L* on μ such that α = β or α is a node-ancestor of β in *G*.

It will be convenient to define:

Definition: μ is a φ-*circular-path* in *G* iff μ is a circular inference / defeat-path in *G* from φ to φ and no defeat-link in *G* can be bypassed.

I will prove the following theorem, which is of central importance in implementing the theory of defeasible reasoning.

Theorem 40: A defeat-link is φ-critical in *G* iff it lies on a φ-circular-path in *G*.

This theorem follows immediately from the next three lemmas.

Lemma 41: If μ_1 and μ_2 are φ-circular-paths and every defeat-link in μ_1 occurs in μ_2, then μ_1 and μ_2 contain the same defeat-links and they occur in the same order.

Proof: Suppose the defeat-links in μ_1 are $\delta_1,...,\delta_n$, occurring in that order. Suppose μ_1 and μ_2 differ first at the *i*th defeat-link. Then μ_1 and μ_2 look as in figure 8. But every defeat-link in μ_1 occurs in μ_2, so δ_i must occur later in μ_2. But then there is a bypass around δ_i* in μ_2, which is impossible if it is a φ-circular-path. ■

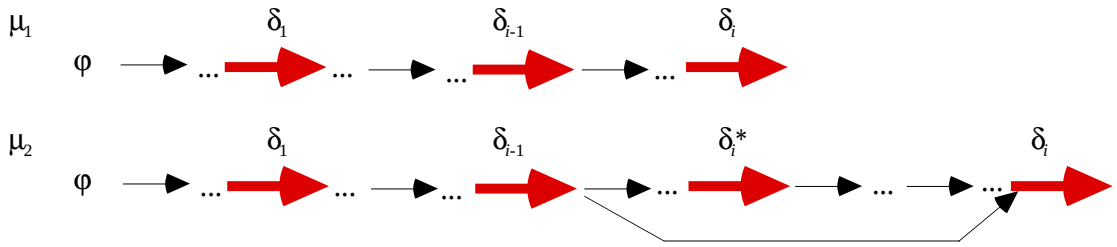


Figure 8. Paths must agree.

Lemma 42: Every defeat-link in a φ-circular-path is φ-critical.

Proof: Suppose δ is a defeat-link on the φ-circular-path μ. Let *D* be the set of all defeaters other than those on μ. If deleting all members of *D* is sufficient to cut all φ-circular-paths not containing δ, then select a minimal subset D_0 of *D* whose deletion is sufficient to cut all φ-circular-paths not containing δ. Adding δ to D_0 gives us a set of defeat-links whose deletion is sufficient to cut all φ-circular-paths. Furtheremore, it is minimal, because adding δ cannot cut any paths not containing δ, and all members of D_0 are required to cut those paths. Thus δ is a member of a minimal set of defeat-links the deletion of which is sufficient to to cut all φ-circular-paths, i.e., δ is φ-critical.

Thus if δ is not φ-critical, there is a φ-circular-path ν not containing δ and not cut by cutting all defeat-links not in μ. That is only possible if every defeat-link in ν is in μ. But then by the previous lemma, μ and ν must contain the same defeat-links, so contrary to supposition δ is in ν. So the supposition that δ is not φ-critical is inconsistent with the supposition that it lies on a φ-circular-path. ■

Lemma 43: If a defeat-link does not occur on any φ-circular-path then it is not φ-critical.

Proof: For every circular inference/defeat-path μ from φ to φ there is a φ-circular-path ν such that every defeat-link in ν is in μ. ν results from removing bypassed defeat-links and support-links in μ and replacing them by their bypasses. It follows that any set of deletions of defeat-links that will cut all φ-circular-paths will also cut every circular inference/defeat-path from φ to φ. Conversely, φ-circular-paths are also circular-paths from φ to φ, so any set of deletions that cuts all circular-paths from φ to φ will also cut all φ-circular-paths. So the φ-circular-paths and the circular-paths from φ to φ have the same sets of deletions of defeat-links sufficient to cut them, and hence the same minimal sets of deletions. If a defeat-link δ does not occur on any φ-circular-path, then it is irrelevant to cutting all the φ-circular-paths, and hence it is not in any minimal set of deletions sufficient to cut all circular-paths from φ to φ, i.e., it is not φ-critical. ■

Theorem 40 follows immediately from lemmas 42 and 43.

A further simplification results from observing that, for the purpose of deciding whether a defeat-link is φ-critical, all we have to know about φ-circular-paths is what defeat-links occur in them. It makes no difference what support-links they contain. So let us define:

Definition: A φ-*defeat-loop* is a sequence μ of defeat-links for which there is a φ-circular-path

ν such that the same defeat-links occur in μ and ν and in the same order.

In other words, to construct a φ-defeat-loop from a φ-circular-path we simply remove all the support-links. We have the following very simple characterization of φ-defeat-loops:

Theorem 44: A sequence $\langle\delta_1,...,\delta_n\rangle$ of defeat-links is a φ-defeat-loop iff (1) φ is a node-ancestor of the root of δ_1 but not of the root of any δ_k for $k > 1$, (2) φ is the target of δ_n, and (3) for each $k < n$, the target of δ_k is equal to or an ancestor of the root of δ_{k+1}, but not of the root of δ_{k+j} for $j > 1$.

The significance of φ-defeat-loops is that by omitting the support-links we make them easier to process, but we stilll have the simple theorem:

Theorem 45: A defeat-link is φ-critical in G iff it lies on a φ-defeat-loop in G.

11.2 A Recursive Computation

Recall that (DJ) was formulated as follows:

(DJ) If ψ is inferred from the basis $\{B_1,...,B_n\}$ in an inference-graph G in accordance with a reason-scheme of strength ρ, $D_1,...,D_k$ are the ψ-independent defeaters for ψ, $D_{k+1},...,D_m$ are the P-dependent defeaters of ψ, then
$j(\psi,G) = \min\{\rho,j(B_1,G),...,j(B_n,G)\} \sim [\max\{j(D_1,G),...,j(D_k,G)\} + \max\{j(D_{k+1},G_P),...,j(D_m,G_P)\}]$.

In most of the cases we have considered, G_φ is an inference-graph in which no node ψ has a ψ-dependent defeater. Thus $j(\psi,G_\varphi)$ can be computed using just (DJ1) in G_φ. This is equivalent to applying (DJ1) in G but ignoring φ-critical defeat-links:

> If ψ is inferred from the basis $\{B_1,...,B_n\}$ in the inference-graph G in accordance with a reason-scheme of strength ρ, $D_1,...,D_k$ are the defeaters for ψ in G that are not φ-critical, and no D_i is ψ-dependent, then
> $j(\psi,G_\varphi) = \min\{\rho,j(B_1,G),...,j(B_n,G)\} \sim \max\{j(D_1,G),...,j(D_k,G)\}$.

So for this computation, it is unnecessary to modify the inference-graph.

In those cases in which there is a node ψ having a ψ-dependent defeater in G_φ, we must instead apply (DJ2):

> If ψ has ψ-dependent defeaters $D_1,...,D_k$ in G_φ then
> $j(\psi,G_\varphi) = j(\psi,(G_\varphi)_\psi) \sim \max\{j(D_1,(G_\varphi)_\psi),...,j(D_k,(G_\varphi)_\psi)\}$.

Applying this computation recursively leads to a sequence of inference-graphs of the form $G_{\varphi_1,...,\varphi_n} = (...(G_{\varphi_n})_{\varphi_{n-1}} ...)_{\varphi_1}$. The inference-graph G is finite because it represents the actual state of an agent's reasoning. It follows that the computation eventually terminates because the inference-graph G has only finitely many defeat-links, and each subsequent $G_{\varphi_1,...,\varphi_n}$ has fewer defeat-links than its predecessor. The sequence of inference-graphs constructed in this way can be characterized

as follows. Where $\varphi_1,...,\varphi_n$ are nodes of an inference-graph G, define recursively:

Definition:

G_φ results from deleting all φ-critical defeat-links from G and making all nodes θ that are either members of the node-basis of φ or φ-independent in G initial with $j(\theta,G_\varphi) = j(\theta,G)$;

$G_{\varphi_1,...,\varphi_n}$ results from deleting all φ_1-critical defeat-links from $G_{\varphi_2,...,\varphi_n}$ and making all nodes θ that are either members of the node-basis of φ_1 or φ_1-independent in $G_{\varphi_2,...,\varphi_n}$ initial with $j(\theta,G_{\varphi_1,...,\varphi_n}) = j(\theta,G_{\varphi_2,...,\varphi_n})$.

To reformulate the recursion so as to avoid constructing modified inference-graphs, we define some new concepts. Where $\varphi_1,...,\varphi_n$ are nodes of an inference-graph G, define recursively:

Definition:

A defeat-link δ of G is $\langle\varphi_1,...,\varphi_n\rangle$-*critical* in G iff (1) δ lies on φ_1-defeat-loop μ in G from containing no $\langle\varphi_2,...,\varphi_n\rangle$-critical defeat-links.

A defeat-link δ of G is *hereditarily*-$\langle\varphi_1,...,\varphi_n\rangle$-*critical* in G iff either δ is $\langle\varphi_1,...,\varphi_n\rangle$-*critical* in G or δ is hereditarily-$\langle\varphi_2,...,\varphi_n\rangle$-*critical* in G.

A defeater (i.e., a node) of G is *hereditarily*-$\langle\varphi_1,...,\varphi_n\rangle$-*critical* in G iff it is the root of a hereditarily-$\langle\varphi_1,...,\varphi_n\rangle$-critical defeat-link in G.

Obviously:

Theorem 46: δ is hereditarily-$\langle\varphi_1,...,\varphi_n\rangle$-critical in G iff δ is φ_1-critical in $G_{\varphi_2,...,\varphi_n}$ or φ_2-critical in $G_{\varphi_2,...,\varphi_n}$ or ... or φ_n-critical in G_{φ_n}.

A defeat-link that is φ_i-critical in $G_{\varphi_i,...,\varphi_n}$ does not exist in $G_{\varphi_j,...,\varphi_n}$ for $j < i$, so:

Theorem 47: δ is φ_1-critical in $G_{\varphi_2,...,\varphi_n}$ iff δ is $\langle\varphi_1,...,\varphi_n\rangle$-critical in G.

Where $\theta,\varphi_1,...,\varphi_n$ are nodes of an inference-graph G, define:

Definition:

θ is $\langle\varphi\rangle$-*independent* of ψ in G iff there is no inference / defeat-path in G from φ to θ.

θ is $\langle\varphi_1,...,\varphi_n\rangle$-*independent* in G iff every inference / defeat-path in G from φ_1 to θ contains a hereditarily-$\langle\varphi_2,...,\varphi_n\rangle$-critical defeat-link.

Theorem 48: θ is $\langle\varphi_1,...,\varphi_n\rangle$-*independent* in G iff θ is φ_1-independent in $G_{\varphi_2,...,\varphi_n}$.

Where ψ is an initial node in G, let $j_0(\psi,G)$ be its assigned value. Let us define recursively:

Definition:

(a) If ψ is initial in G then $j_{\varphi_1,...,\varphi_n}(\psi,G) = j_0(\psi,G)$;

(b) If ψ is $\langle\varphi_1,...,\varphi_n\rangle$-*independent* in G then $j_{\varphi_1,...,\varphi_n}(\psi,G) = j_{\varphi_2,...,\varphi_n}(\psi,G)$;

(c) If ψ is inferred from the basis $\{B_1,...,B_n\}$ in an inference-graph G in accordance with a reason-scheme of strength ρ, $D_1,...,D_k$ are the defeaters for ψ that are $\langle\varphi_1,...,\varphi_n\rangle$-independent of ψ in G and are not hereditarily-$\langle\varphi_1,...,\varphi_n\rangle$-critical in G, and $D_{k+1},...,D_m$ are the defeaters for ψ that are $\langle\varphi_1,...,\varphi_n\rangle$-dependent on ψ in G and are not hereditarily-$\langle\varphi_1,...,\varphi_n\rangle$-critical in G, then

$$j_{\varphi_1,...,\varphi_n}(\psi,G) = \min\{\rho, j_{\varphi_1,...,\varphi_n}(B_1,G),...,j_{\varphi_1,...,\varphi_n}(B_n,G)\} \sim [\max\{j_{\varphi_1,...,\varphi_n}(D_1,G),...,j_{\varphi_1,...,\varphi_n}(D_k,G)\} + \max\{j_{\psi,\varphi_1,...,\varphi_n}(D_{k+1},G),...,j_{\psi,\varphi_1,...,\varphi_n}(D_m,G)\}].$$

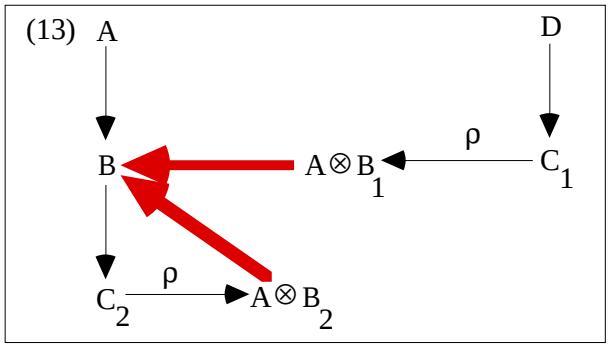
The reason this is a recursive definition is that we always reach an n at which there are no more $\langle\varphi_1,...,\varphi_n\rangle$-dependent defeaters, and then the values of all nodes are computed recursively in terms of the values assigned to initial nodes.

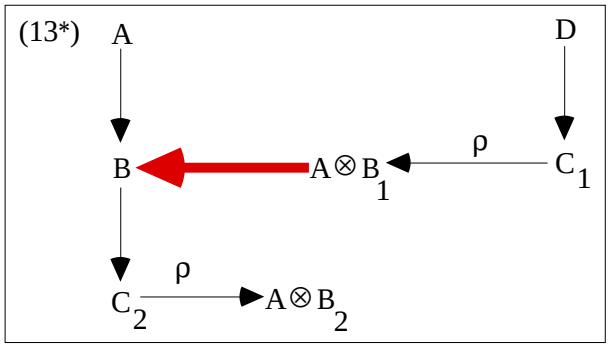
It is now trivial to prove by induction on n that:

Theorem 49: $j(\psi,G_{\varphi_1,...,\varphi_n}) = j_{\varphi_1,...,\varphi_n}(\psi,G)$.

Thus we have a recursive definition of the degree of justification of a node that that computes the degrees of justification entirely by reference to the given inference-graph rather than by building a sequence of modified inference-graphs in accordance with the original analysis.

To illustrate the computation provided by theorem 49, consider inference-graph (13). Here we have two separate arguments for C, from which $A\otimes B$ is inferred with reason-strength ρ. Nodes supporting the same conclusions are distinguished by subscripting the conclusions. To compute $j(B,G_{13})$ using (DJ1) and (DJ2), we remove the B-critical link to get (13*). Then





$j(B,G_{13}) = j_0(A,G_{13}) \sim [j(A\otimes B_1,G_{13}) + j(A\otimes B_2,G_{13^*})]$.

$j(A\otimes B_1,G_{13}) = \min\{\rho,j(C_1,G_{13})\} = \min\{\rho,j_0(D,G_{13})\}$. $j(A\otimes B_2,G_{13^*}) = \min\{\rho,j(C_2,G_{13^*})\} = \min\{\rho,j(B,G_{13^*})\} = \min\{\rho, (j_0(A,G_{13}) \sim \min\{\rho,j_0(D,G_{13})\})\}$.

Thus

$j(B,G_{13}) = j_0(A,G_{13}) \sim [\min\{\rho,j_0(D,G_{13})\} + \min\{\rho, (j_0(A,G_{13}) \sim \min\{\rho,j_0(D,G_{13})\})\}]$.

For example, if $j_0(A,G_{13}) = j_0(D,G_{13})$, and $\rho < .5 \cdot j_0(A,G_{13})$. Then $j(B) = j_0(A,G_{13}) \sim 2\rho$.

To repeat the computation using theorem 49,

$$j(B,G_{13}) = j_0(A,G_{13}) \sim [j(A \otimes B_1,G_{13}) + j_B(A \otimes B_2,G_{13})].$$

$$j(A \otimes B_1,G_{13}) = \min\{\rho,j(C_1,G_{13})\} = \min\{\rho,j_0(D,G_{13})\}.$$

$$j_B(A \otimes B_2,G_{13}) = \min\{\rho,j_B(C_2,G_{13})\} = \min\{\rho,j_B(B,G_{13})\}.$$

$A \otimes B_1$ is the only B-independent defeater of B, and it is not B-critical. Thus

$$j_B(B,G_{13}) = j_B(A,G_{13}) \sim j_B(A \otimes B_1,G_{13}) = j_0(A,G_{13}) \sim j(A \otimes B_1,G_{13}) = j_0(A,G_{13}) \sim \min\{\rho,j_0(D,G_{13})\}.$$

Therefore, $j_B(A \otimes B_2,G_{13}) = \min\{\rho, j_0(A,G_{13}) \sim \min\{\rho,j_0(D,G_{13})\}\}$.

Hence once again,

$$j(B,G_{13}) = j_0(A,G_{13}) \sim [\min\{\rho,j_0(D,G_{13})\} + \min\{\rho, j_0(A,G_{13}) \sim \min\{\rho,j_0(D,G_{13})\}\}].$$

Theorem 49 constitutes the desired recursive characterization of $j(\varphi,G)$. This could be implemented straightforwardly. However, as the next section shows, further improvements are possible.

12. Inference-Hypergraphs

12.1 Two Kinds of Inference-Graphs

In the interest of theoretical clarity, inference-graphs were defined in such a way that different arguments for the same conclusion are represented by different nodes. This made it clearer how the algorithm for computing degrees of justification works. However, for the purpose of implementing defeasible reasoning, using different nodes to represent different arguments for the same conclusion is an inefficient representation, because it leads to needless duplication. If we have two arguments supporting a single conclusion, then any further reasoning from that conclusion will generate two different nodes. If we have two arguments for each of two conclusions, and another inference proceeds from those two conclusions, the latter will have to be represented by four different nodes in the inference-graph, and so on. This is illustrated in figure 9, where P and Q are each inferred in two separate ways, and then R is inferred from P and Q.

A more efficient representation of reasoning would take the inference-graph to be a hypergraph rather than a standard graph. In a hypergraph, nodes are linked to *sets* of nodes rather than individual nodes.[17] In an inference-hypergraph, when we have multiple arguments for a conclusion,

---

[17] This observation derives from my (1994). This section adapts the approach taken there to the new computation of degrees of justification. I take the term "hypergraph" from Pearl (1988), who observes that Bayesian nets are

the single node representing that conclusion will be tied to different bases by separate links. This is illustrated in figure 10 by an inference-hypergraph encoding the same reasoning as the standard inference-graph in figure 9. In an inference-hypergraph, the root of a support-link is its *basis*, i.e., the set of earlier conclusions from which the inference is made.

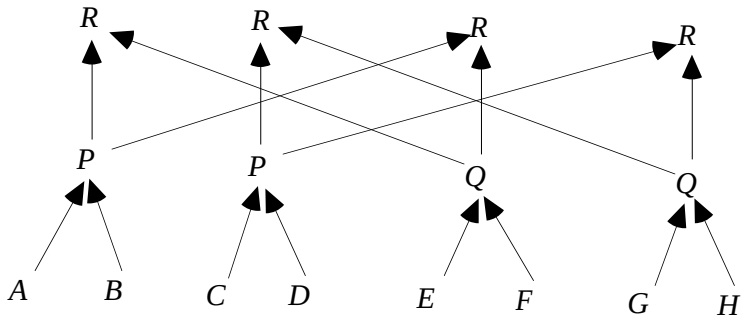


Figure 9. Inference-graph with multiple arguments for a single conclusion.

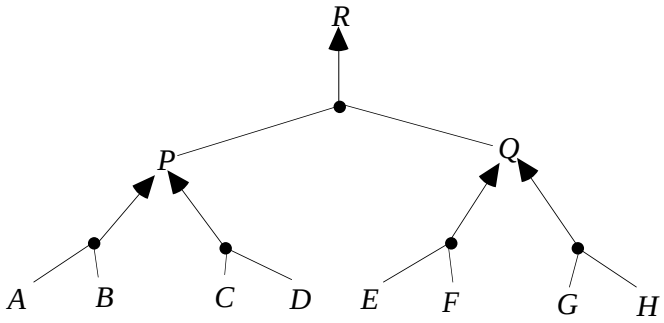


Figure 10. An inference-hypergraph

Although hypergraphs provide an efficient representation of reasoning, they complicate the computation of degrees of justification. Using simple inference-graphs, we can use principles (DJ1) and (DJ2) to compute degrees of justification. If we are to use hypergraphs in the implementation, we must find a computation for hypergraphs that is equivalent to that for simple inference-graphs. A simple inference-graph can be rewritten as a hypergraph in which each node of the hypergraph corresponds to a set of nodes of the simple graph. A node of the simple inference-graph corresponds to an *argument* in the hypergraph. An argument is a kind of connected sub-tree of the graph. More precisely:

Definition: An *argument in* an inference-hypergraph *HG for* a node φ is a minimal subset A of the nodes and support-links of the graph such that (1) if a node in A has any support-links in *HG*, exactly one of them is in A, (2) if a support-link is in A then the nodes in its

---

hypergraphs. I previously called hypergraphs "and/or graphs".

support-link-basis are also in A, and (3) φ is in A.

Nodes in the simple inference-graph correspond one-one to arguments in the inference-hypergraph.

12.2 Critical Links in Hypergraphs

It was argued in section five that the degree of justification of a conclusion should be the maximum of the degrees of justification supplied by the different arguments supporting that conclusion. Thus the result we want for inference-hypergraphs is the following *Correspondence Principle*:

> The degree of justification of a node of the inference-hypergraph is equal to the maximum of the degrees of justification of the corresponding nodes of the simple inference-graph.

To accomplish this, it helps to assign degrees of justification to support-links and defeat-links as well as to nodes. Some nodes in an inference-hypergraph HG will be initial, in which case they are simply assigned a degree of justification $j_0(P,HG)$. I assume that such nodes have no support-links. If a node is not initial, its degree of justification is the maximum of the degrees of justification of its support-links. If a node is not initial and it has no support-links, we stipulate that its degree of justification is zero. This cannot happen naturally, but certain constructions that occur in the computation of degrees of justification can produce inference-hypergraphs having such nodes. Normally, the degree of justification of a defeat-link will be the same as the degree of justification of its root, but again, we will introduce a construction below in which they sometimes differ. Having characterized the degree of justification of a node of the inference-graph in terms of the degrees of justification of its support-links, the remaining problem is how to compute the degrees of justification of support-links.

Let us make a clear distinction between defeaters and defeat-links. Defeaters will be taken to be the inference-nodes that are the roots of defeat-links, and defeat-links relate defeaters to what they defeat. In an inference-hypergraph, undercutting defeat-links attach to support-links rather than nodes. A rebutting defeat-link could be regarded as attaching to either a node or to all defeasible support-links for the node. It will be more convenient to adopt the latter convention so that both undercutting and rebutting defeaters can be treated uniformly. Where λ is a support-link, let ⊗λ be its undercutting defeater (the inference-node that is the root of the defeat-link) and let ~λ be its rebutting defeater. That is, if the root of λ is $\{B_1,...,B_n\}$ and the target is φ, ⊗λ is $[(B_1\&...\&B_n) \otimes \varphi]$ and ~λ is ~φ. Let ◄⊗λ► and ◄~λ► be the defeat-links corresponding to the defeaters. Conversely, where δ is a defeat-link, let Δ(δ) be the defeater that is its root. So δ = ◄Δ(δ)►. In inference-hypergraphs, the target of a defeat-link is a support-link rather than an inference-node. It will be convenient to define the *ultimate-target* of the defeat-link to be the target of that support-link, i.e., the target of the target of the defeat-link.

Let us say that a simple graph *corresponds* to a hypergraph iff there is a many-one mapping from the nodes of the simple graph to the nodes of the hypergraph which relates all nodes of the simple inference-graph supporting a proposition φ to the single node of the hypergraph supporting φ, and preserves support relations and defeat relations. Representing links by ordered pairs, we can define precisely:

Definition:

A simple inference-graph G *corresponds to* an inference-hypergraph HG iff there is a function μ mapping the nodes of G onto the nodes of HG such that:

(a) if α is a node of G and β is a node of HG, $\mu(\alpha) = \beta$ iff α and β support the same proposition;

(b) if α is a node of G and $\Gamma = \{\beta \mid \langle\beta,\alpha\rangle$ is a support-link for α in $G\}$ then $\langle\{\mu(\gamma) \mid \gamma\in\Gamma\},\mu(\alpha)\rangle$ is a support-link in HG;

(c) if $\langle\Gamma,\alpha\rangle$ is a support-link in HG then for all nodes β,γ in G, if $\mu(\beta)\in\Gamma$ and $\mu(\gamma) = \alpha$ then $\langle\beta,\gamma\rangle$ is a support-link in G;

(d) $\langle\beta,\alpha\rangle$ is a defeat-link in G iff, if $\langle\gamma_1,\alpha\rangle,...,\langle\gamma_n,\alpha\rangle$ are the support-links for α in G then $\langle\mu(\beta),\langle\{\mu(\gamma_1),...,\mu(\gamma_n)\},\mu(\alpha)\rangle\rangle$ is a defeat-link in HG.

Where x is an item in the inference hypergraph (i.e., a support-link, defeat-link, or inference-node), the corresponding items in the simple inference-graph will be called *the instances of x*. To make this precise, let us extend the many-one mapping μ to defeat-links and support-links:

Definition:

If μ is a many-one mapping from the nodes of G to the nodes of HG:

(a) if $\langle\beta,\alpha\rangle$ is a support-link in G then $\mu(\langle\beta,\alpha\rangle) = \langle\{\mu(\gamma) \mid \langle\gamma,\alpha\rangle$ is a support-link in $G\},\mu(\alpha)\rangle$;

(b) if $\langle\beta,\alpha\rangle$ is a defeat-link in G then $\mu(\langle\beta,\alpha\rangle) = \langle\mu(\beta),\langle\{\mu(\gamma_1),...,\mu(\gamma_n)\},\mu(\alpha)\rangle\rangle$ where $\langle\gamma_1,\alpha\rangle,...,\langle\gamma_n,\alpha\rangle$ are the support-links for α in G.

Then if x is an inference-node, support-link, or defeat-link in G, $\mu(x)$ is the corresponding item in HG, and x is an instance of $\mu(x)$. An item in a hypergraph represents the set of all its instances in the simple graph.

In inference-hypergraphs, we can define inference/defeat-paths pretty much as we did for simple inference-graphs, except that they are paths from support-links to support-links rather than from nodes to nodes:

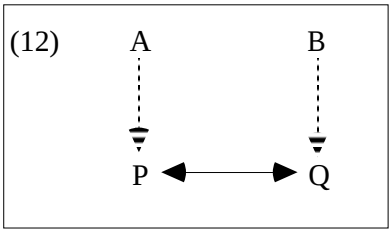
Definition: An *inference/defeat-path* from a support-link γ to a support-link or defeat-link λ in an inference-hypergraph HG is a sequence of support-links and defeat-links in HG such that (1) the first link is γ; (2) the last link in the path is λ; (3) the root of each defeat-link after the first member of the path is the target of the preceding link; (4) some member of the basis of of each support-link after the first member of the path is the ultimate-target of the preceding link; and (5) the path does not contain an internal loop, i.e., no two links in the path have the same tultimate-arget.

Definition: A *circular inference/defeat-path* from a support-link λ to itself is an inference/defeat-path from λ to a defeat-link for λ.

Definition: If λ is a support-link, a support-link or defeat-link γ is λ-independent in an inference-hypergraph HG iff there is no inference/defeat-path from λ to γ in HG.

Definition: If λ is a support-link, a node θ is λ-independent in an inference-hypergraph *HG* iff there is no inference / defeat-path from λ to a support-link for θ.

Given an inference / defeat-path in the simple graph, the corresponding sequence of links in the hypergraph is a corresponding inference / defeat-path in the hypergraph. Furthermore, given an inference / defeat-path in the hypergraph, every corresponding path (i.e., every instance of it) in the simple graph is an inference / defeat-path.



Inference-paths must be defined somewhat differently in inference-hypergraphs than in simple inference-graphs. The difficulty is that if we simply define inference-paths to be sets of linked support-links, they can be circular. E.g., suppose *P* and *Q* are logically equivalent. Then if we have independent reasons for *P* and for *Q*, we can derive each from the other. This makes perfectly good sense. If the independent argument for *P* is subsequently defeated, the argument that derives *P* from *Q* will still support *P* as long as the argument supporting *Q* is not defeated, and vice versa. Thus we can have an inference-hypergraph like (12) with "inference-loops". These are inference-hypergraphs in which some node is an inference-descendant of itself (where the inference-descendants of a node are those nodes that can be reached by following sequences of support-links). In this inference-graph, we do not want to count paths like $A \rightarrow P \rightarrow Q \rightarrow P$ as inference-paths. We can handle this by defining:

Definition: An *inference-path* from a node φ to a node θ is a sequence of support-links such that (1) φ is a member of the root of the first link, (2) the target of the last is θ, (3) some member of each support-link after the first is the target of the preceding link, and (4) the path is non-circular, i.e., $\varphi \neq \theta$ and no two links in the path have the same target.

Definition: An inference-node α is a node-ancestor of an inference-node β iff there is an inference-path from α to β.

Then we can define, for hypergraphs, where λ is a support-link:

Definition: A defeat-link δ is λ-*critical* in *HG* iff δ is a member of a minimal set *D* of defeat-links such that the deletion of all members of *D* from *HG* is sufficient to cut all circular inference / defeat-paths in *HG* from λ to λ.

As before, we define:

Definition: Where μ is a circular inference / defeat-path in *HG* from λ to λ, δ *can be bypassed*

on μ iff there is no node α preceding the root of δ on μ and node β preceded by the ultimate-target of δ on μ such that α = β or α is a node-ancestor of β in *HG*.

Definition: μ is a λ-*circular inference/defeat-path* in *HG* iff μ is a circular inference / defeat-path in *HG* from λ to λ and no defeat-link in μ can be bypassed.

Definition: A sequence $\langle \delta_1,...,\delta_n \rangle$ of defeat-links is a λ-*defeat-loop* iff (1) the target of λ is a node-ancestor of the root of δ_1 but not of the root of any δ_k for $k > 1$, (2) λ is the target of δ_n, and (3) for each $k < n$, the ultimate-target of δ_k is equal to or an ancestor of the root of δ_{k+1}, but not of the root of δ_{k+j} for $j > 1$.

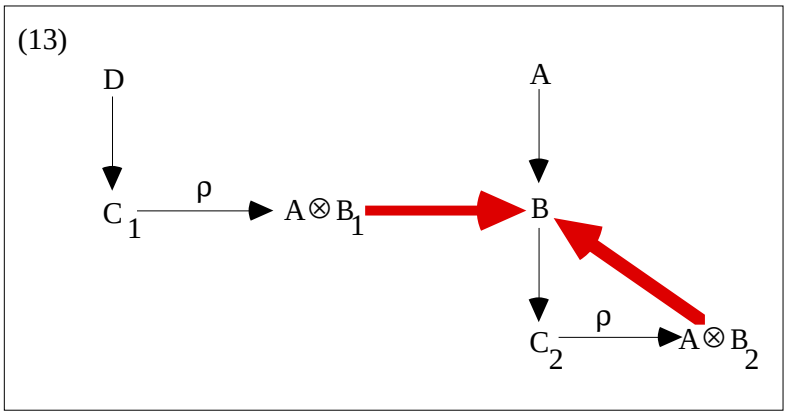
Then once again we have:

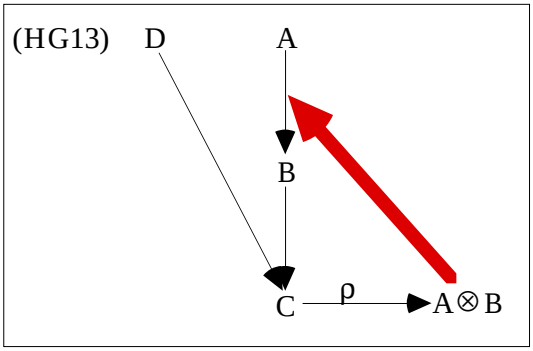
Theorem 50: A defeat-link is λ-critical in *HG* iff it lies on a λ-circular-path in *HG*.

Theorem 51: A defeat-link is λ-critical in *HG* iff it lies on a λ-defeat-loop in *HG*.

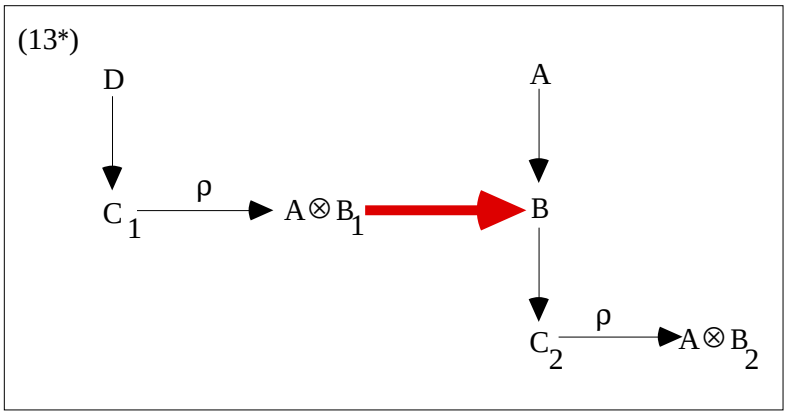
12.3 The Problem of Multiple Instances

Relating inference-hypergraphs to simple inference-graphs is complicated by the fact that a defeat-link in an inference-hypergraph can correspond to several different defeat-links in the simple inference-graph. Where *HG* is a hypergraph and *G* the corresponding simple graph, the instances of a defeat-link δ in *HG* are the defeat-links in *G* whose roots are instances of the root Δ(δ) of δ. In talking about the relationship between a hypergraph *HG* and the corresponding simple graph *G*, it will be convenient to allow ourselves uniform terminology. Where λ is a support-link in *HG*, we talk about a defeat-link being λ-critical. The corresponding notion for simple inference-graphs is *Q*-criticality, where *Q* is the target of λ. But I will sometimes talk about an instance of the defeat-link being λ-critical in *G*, where that is short for saying that it is *Q*-critical. Similarly, I will sometimes talk about nodes of the simple graph being λ-independent when they are *Q*-independent.





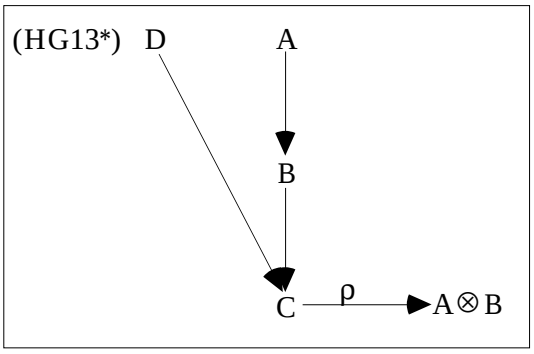
To illustrate the problem of a defeat-link in a hypergraph having multiple instances in a simple graph, consider the simple inference-graph (13) again and the corresponding inference-hypergraph $HG13$. Here ρ is the reason-strength for the inference from C to $A\otimes B$. Writing both support-links and defeat-links as ordered pairs, the defeat-link $\langle A\otimes B,\langle A,B\rangle\rangle$ in $HG13$ corresponds to the two defeat-links $\langle A\otimes B_1,B\rangle$ and $\langle A\otimes B_2,B\rangle$ in (13). In general, if a defeat-link δ is λ-critical in the hypergraph, then *some* instance of it is λ-critical in the simple graph. More precisely, if δ is λ-critical in the hypergraph by virtue of lying on the inference / defeat-path μ, then for every corresponding inference / defeat-path Σ in the simple graph, the instance of δ that lies on Σ is λ-critical in the simple graph. However, there can also be instances of δ in the simple graph that do not lie on inference / defeat-paths corresponding to μ. For example, $\langle A\otimes B,\langle A,B\rangle\rangle$ is $\langle A,B\rangle$-critical in ($HG13$), but only $\langle A\otimes B_2,B\rangle$, not $\langle A\otimes B_1,B\rangle$, is B-critical in (13). $\langle A\otimes B_1,B\rangle$ does not lie on an B-circular-path. Because $A\otimes B_1$ and $A\otimes B_2$ must be represented by a single node $A\otimes B$ in the hypergraph, there is no way to represent the difference between $A\otimes B_1$ and $A\otimes B_2$.



Our objective is to find a way of computing degrees of justification in the inference-hypergraph that agrees with the computation in the simple inference-graph. To compute $j(B,G_{13})$ we removed the B-critical link to get (13*), and then computed that $j(B,G_{13}) = j(B,G_{13^*}) \sim j(A\otimes B_2,G_{13^*}) = j(A,G_{13})$ ~ [min{ρ,$j(D,G_{13})$} + min{ρ, ($j(A,G_{13})$ ~ min{ρ,$j(D,G_{13})$})}]. For example, if $j(A,G_{13}) = j(D,G_{13})$, and ρ < .5·$j(A,G_{13})$, then $j(B,G_{13}) = j(A,G_{13})$ ~ 2ρ. It might be supposed that we can compute degrees of justification analogously in the inference-hypergraph by deleting $\langle A,B\rangle$-critical defeat-links. However, this does not work. Removing $\langle A,B\rangle$-critical defeat-links would produce inference-graph $HG13^*$. The computation analogous to that employed in (13) would yield

$j(B,HG13) = j(B,HG13^*) \sim j(A \otimes B,HG13^*)$
$= j(A,HG13) \sim \min\{\rho,j(C,HG13^*)\}$
$= j(A,HG13) \sim \min\{\rho,\max\{j(A,HG13),j(D,HG13)\}\}$.

If $\rho < j(A,HG13) = j(D,HG13)$ then $j(B,HG13) = j(A,HG13) \sim \rho$. So this does not validate the Correspondence Principle.



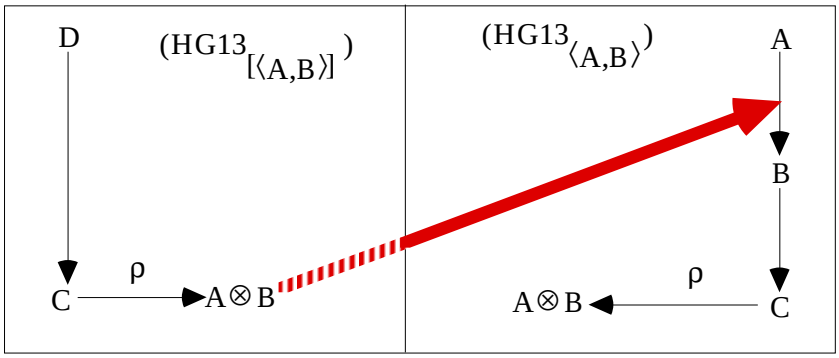
This illustrates a general point. Nodes in a simple inference-graph correspond to arguments in the hypergraph. In an inference-hypergraph *HG*, if a defeat-link δ for a support-link λ with target *P* has multiple arguments supporting its root, some of those arguments may correspond to *P*-critical instances of δ in the corresponding simple inference-graph *G* and some of those arguments may correspond to *P*-noncritical instances. The *P*-critical and *P*-noncritical instances play different roles in the computation of degrees of justifications. In the simple inference-graph this is accommodated by the fact that the different defeat-links have different roots and the critical defeat-link can be deleted (detached from its root) without the noncritical defeat-link being detached from its root. But in a hypergraph, the roots of these defeat-links must all be represented by the same inference-node, because there cannot be multiple inference-nodes for the same proposition Δ(δ), and correspondingly there can only be one defeat-link in the hypergraph. It is thus impossible to detach critical defeat-links from their roots without detaching noncritical defeat-links from theirs.

12.4 Splitting the Hypergraph

One way of solving this problem is to split the initial hypergraph *HG* into two hypergraphs, with Δ(δ) serving as the root of critical links in one of the hypergraphs and serving as the root of noncritical links in the other. Let HG_{λ} be the hypergraph in which, for a defeat-link δ, Δ(δ) represents the roots of the λ-critical instances of δ in *G*. Let $HG_{[\lambda]}$ be the hypergraph in which Δ(δ) represents the roots of the λ-noncritical instances of δ in *G*. When δ is λ-critical in *HG*, there are two ways that an instance of δ may fail to be λ-critical in *G*. First, it could fail to lie on any circular inference / defeat-path from λ to λ. Then either there is no inference / defeat path from that instance of δ to λ in *G*, in which case it is irrelevant to the computation of the degree of justification of λ and we can omit it from the two hypergraphs, or there is no inference / defeat path from λ to the instance of δ, i.e., the instance of δ is λ-independent. The latter is what happens in inference-graph (13). Second, the instance of δ could lie on circular inference / defeat-paths from λ to λ in *G* but be bypassed on all of them. I will give an example of this below.

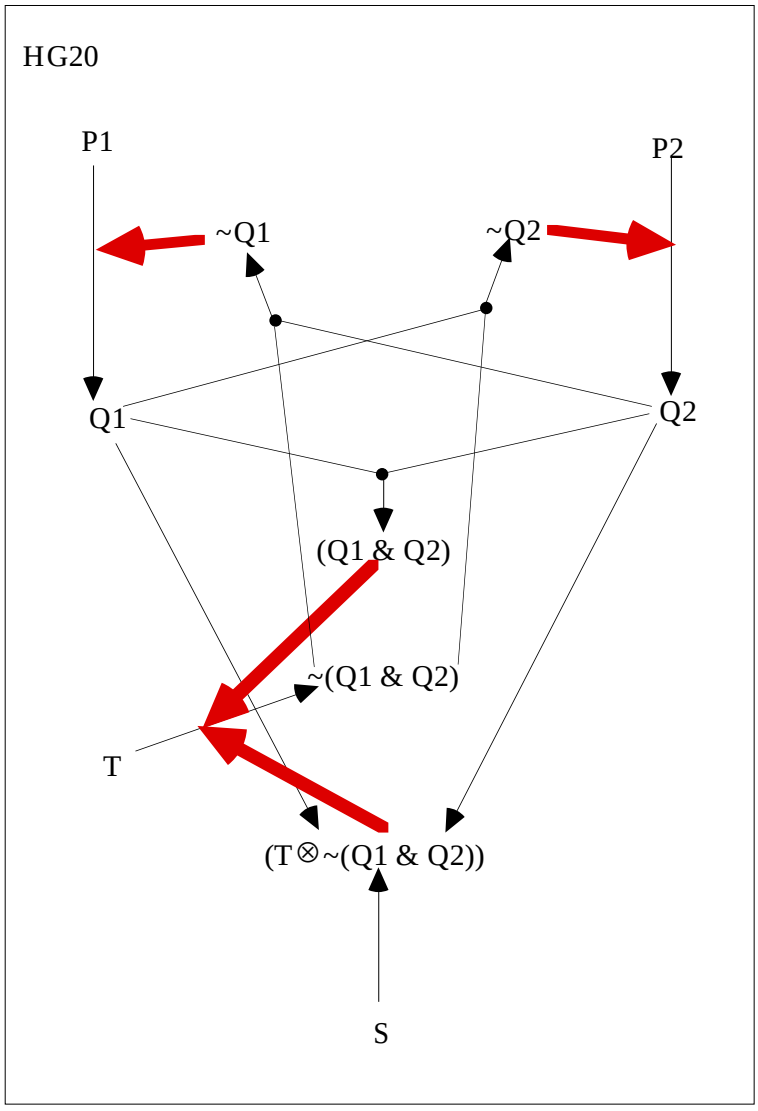
Let us begin with the first case. Suppose we want to compute the degree of justification of a support-link λ in an inference-hypergraph *HG* in which there is a λ-critical defeat-link δ, and let *G* be the corresponding simple inference-graph. Suppose the only λ-noncritical instances of δ in *G* are λ-independent. *HG*13 is an example of this. If we can isolate the λ-dependent and λ-independent arguments for Δ(δ), we can compute both a λ-dependent and a λ-independent degree of justification for the defeat-link, and they will represent the degrees of justification of the two distinct defeaters in *G*.

Using simple inference-graphs we were able to compute the *P*-independent and *P*-dependent values for nodes by modifying the inference-graph. The *P*-independent values were computed in the original inference-graph *G*, and then the *P*-dependent values were computed in the inference-graph G_P. To accomplish the same thing using hypergraphs, we divide the original hypergraph *HG* into the two separate hypergraphs HG_{λ} and $HG_{[\lambda]}$. The first should contain the λ-dependent arguments occurring in *HG*, and the second should contain the λ-independent arguments. To compute the value of λ in HG_{λ}, we must subtract the value of ⊗λ in $HG_{[\lambda]}$ from the value of the basis of λ in HG_{λ}. This amounts to having a defeat link stretching from ⊗λ in $HG_{[\lambda]}$ to λ in HG_{λ}. Thus to compute the justification for ⟨*A*,*B*⟩ in *HG*13, we can construct the hypergraphs $HG13_{\langle A,B\rangle}$ and $HG13_{[\langle A,B\rangle]}$, as diagrammed below. The only difficulty is that a defeat link for a support-link in one inference-graph cannot literally have its root in a different hypergraph. But we can circumvent this difficulty by allowing hypergraphs to have "free floating" defeat-links that are detached from what would normally be their roots. They become analogous to initial nodes in having their degrees of justification stipulated rather than computed.



Formulating these ideas more carefully, we can compute λ-independent values for nodes by deleting arguments containing λ-dependent support-links. Thus in inference-graph *HG*13, the ⟨*A*,*B*⟩-independent strength of *A*⊗*B* can be computed by removing from the inference-graph all support-links that occur only in arguments containing ⟨*A*,*B*⟩-dependent support-links, producing $HG13_{[\langle A,B\rangle]}$. To compute the ⟨*A*,*B*⟩-dependent strength of *A*⊗*B*, we use the ⟨*A*,*B*⟩-independent strength to attenuate the strength of the support-link ⟨*A*,*B*⟩, as in $HG13_{\langle A,B\rangle}$. The latter inference-graph is constructed from *HG*13 by (1) removing ⟨*A*,*B*⟩-independent arguments for *A*⊗*B*, (2) removing ⟨*A*,*B*⟩-critical defeat-links, and (3) detaching the defeat-link ⟨*A*⊗*B*,⟨*A*,*B*⟩⟩ from its root *A*⊗*B* and treating it as an initial defeat-link whose value is inherited from the value for *A*⊗*B* in $HG13_{[\langle A,B\rangle]}$. The strength computed for *A*⊗*B* in $HG13_{\langle A,B\rangle}$ will then be the maximum strength of the ⟨*A*,*B*⟩-dependent defeaters for ⟨*A*,*B*⟩. It follows by (DJ) that $j(B,HG13) = j(A,HG13) \sim [j(A\otimes B,HG13_{[\langle A,B\rangle]}) + j(A\otimes B,HG13_{\langle A,B\rangle})]$. This should be the value computed for $j(\langle A,B\rangle,HG13)$.

Before considering the details of how $\mathcal{HG}13_{\langle A,B\rangle}$ and $\mathcal{HG}13_{[\langle A,B\rangle]}$ are constructed, let us consider an example of the other way in which a λ-critical defeat-link can have λ-noncritical instances. Thus far we have discussed the case of a hypergraph $\mathcal{HG}$ and corresponding simple inference-graph G in which the only λ-noncritical instances of a λ-critical defeat-link δ in $\mathcal{HG}$ are λ-independent in G. However, as we noted above, a λ-critical defeat-link δ in $\mathcal{HG}$ can also have a λ-noncritical instance in G by virtue of lying on a circular inference/defeat-path in G from λ to λ but be bypassed. Again, we want these λ-noncritical instances to have their values computed in $\mathcal{HG}_{[\lambda]}$ and then imported into $\mathcal{HG}_{\lambda}$ in the form of "initial" defeat-links.



For an example of this phenomenon, consider $\mathcal{HG}20$. $\mathcal{HG}20$ is a simplified version of the inference-hypergraph that I have argued elsewhere (my 1995) represents the structure of the paradox of the preface, but with an additional support-link from S to $(T\otimes\sim(Q_1 \,\&\, Q_2))$ added for present purposes. If we suppose that all premises and inference-rules have equal strength, then we should get the result that $j(Q_1) = j(P_1)$, i.e., all of Q_1, Q_2, and $(Q_1 \,\&\, Q_2)$ are justified. (In the paradox of the preface, Q_1 and Q_2 represent the claims made in the book, although that is not

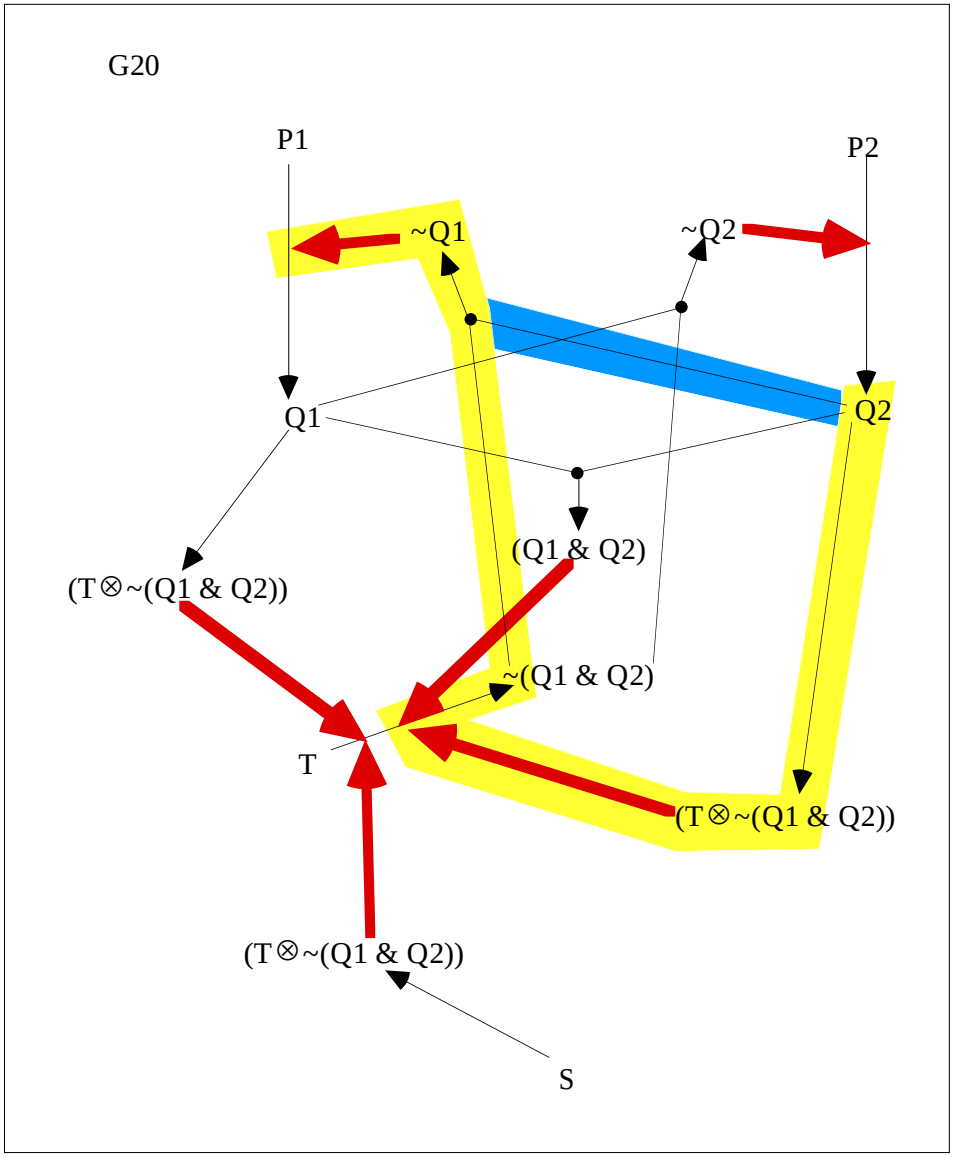
really relevant to the present discussion.) To see that the theory of simple inference-graphs gives us this result, note that *HG*20 corresponds to the simple inference-graph *G*20. What is novel about *G*20 is that it contains three separate arguments for $(T \otimes \sim(Q_1 \,\&\, Q_2))$, and accordingly three separate undercutting defeaters for the support-link $\langle T, \sim(Q_1 \,\&\, Q_2)\rangle$. The leftmost undercutter is Q_1-critical. The bottom undercutter is Q_1-independent. The interesting one is the right undercutter which, although Q_1-dependent, is not Q_1-critical. This is most easily seen by noting that there is only one circular inference / defeat-path from Q_1 to Q_1 that includes the rightmost undercutter, and it includes the subpath

$$\langle Q_2, (T \otimes \sim(Q_1 \,\&\, Q_2))\rangle, \langle (T \otimes \sim(Q_1 \,\&\, Q_2)), \langle T, \sim(Q_1 \,\&\, Q_2)\rangle\rangle, \langle \{\sim(Q_1 \,\&\, Q_2), Q_2\}, \sim Q_1\rangle,$$

which is marked in yellow (or light gray). This subpath is bypassed by the one-step path:

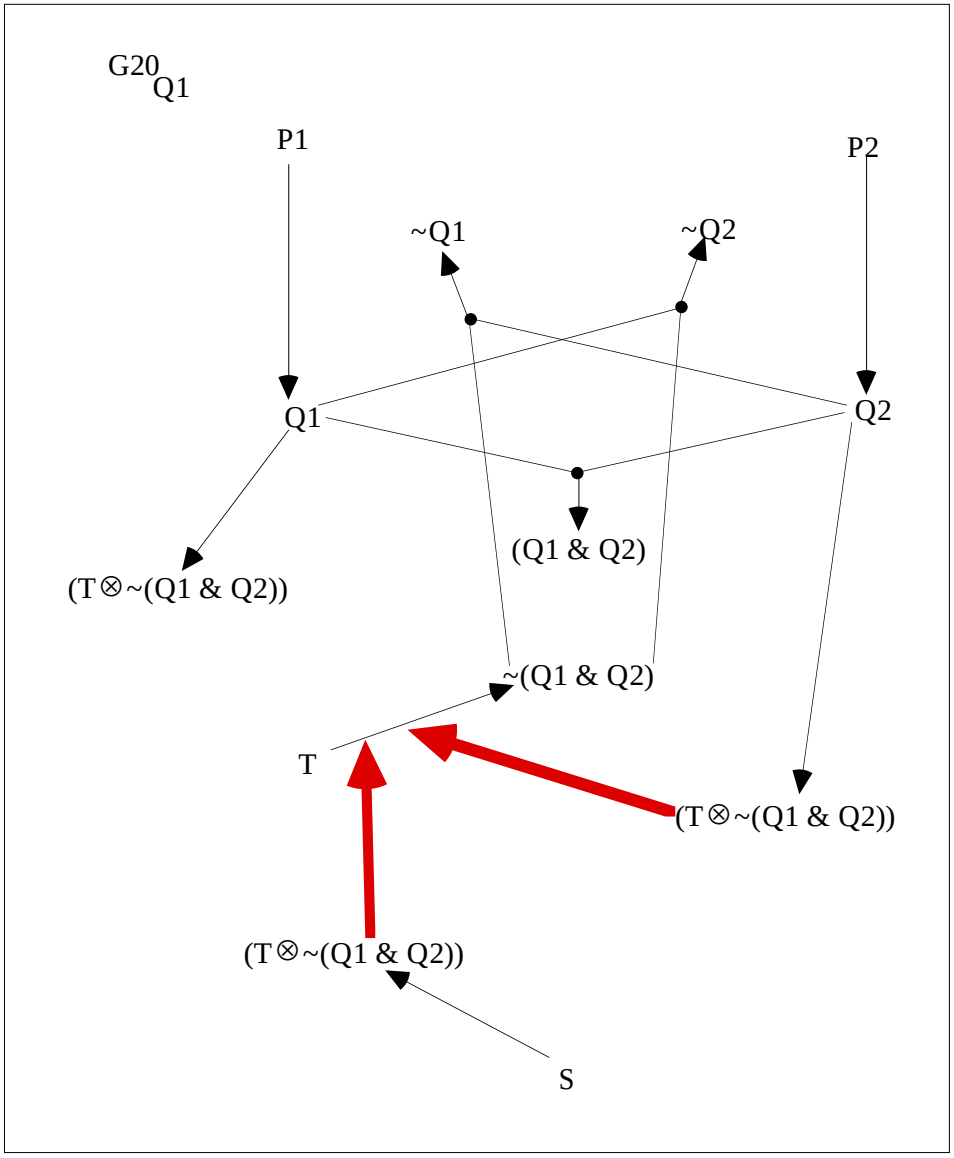
$$\langle \{\sim(Q_1 \,\&\, Q_2), Q_2\}, \sim Q_1\rangle,$$

which is marked in blue (or darker gray). Thus the rightmost defeat-link is not Q_1-critical.



This means that in computing $j(Q_1,G20)$, we construct $G20_{Q1}$ and then calculate:

$$j(Q_1,G20) = j(Q_1,G20_{Q1}) \sim j(\sim Q_1,G20_{Q1}).$$

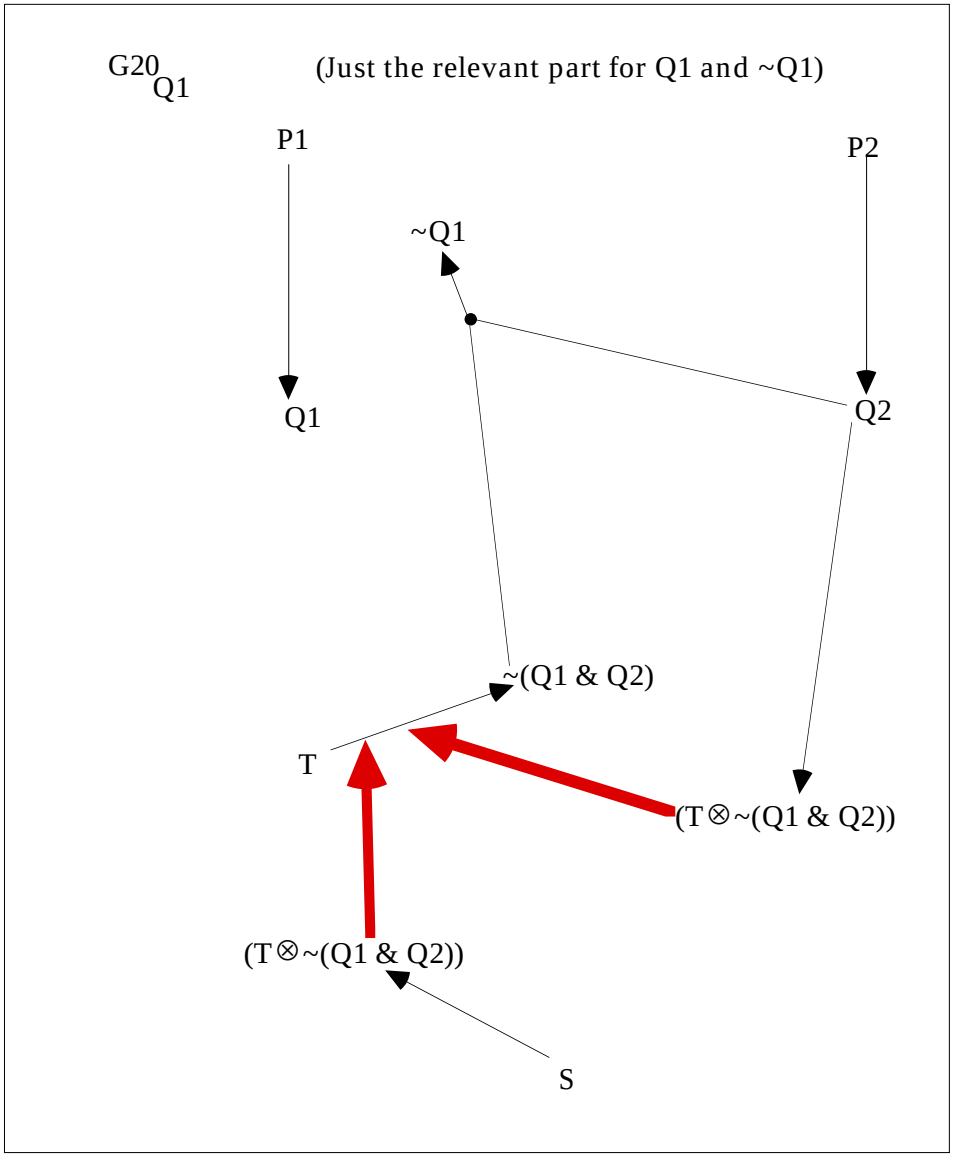


Much of $G20_{Q1}$ is irrelevant to this computation, so I have redrawn the graph below to show just those parts that play a role in the computation. We find that

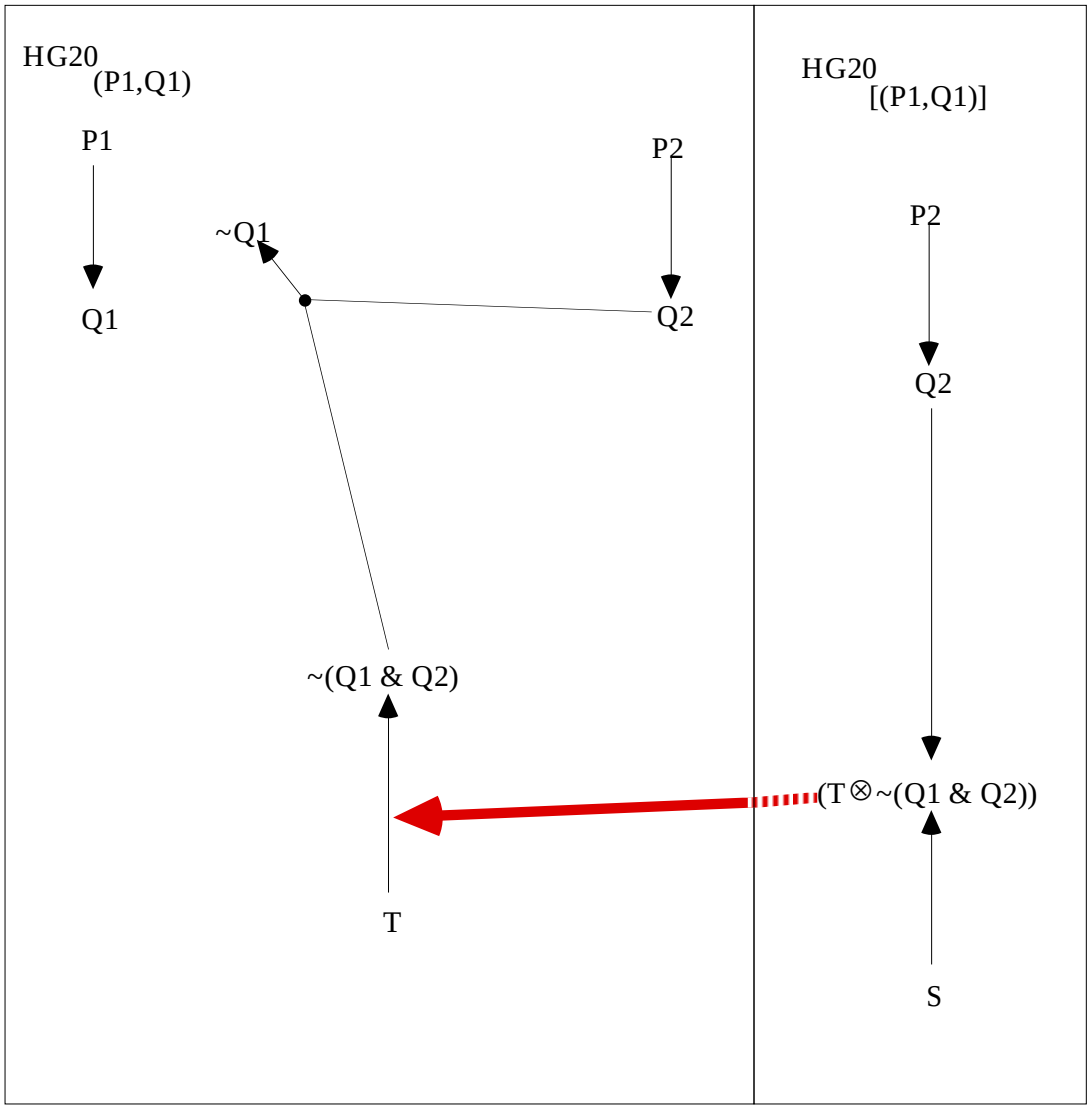
$$j(\sim Q_1, G20_{Q1}) = \min\{ j(Q_2, G20_{Q1}), j(\sim(Q_1 \,\&\, Q_2), G20_{Q1})\}.$$

$$j(\sim(Q_1 \,\&\, Q_2), G20_{Q1}) = j(T, G20_{Q1}) \sim \max\{ j(S, G20_{Q1}), j(Q_2, G20_{Q1})\}.$$

Thus $j(\sim Q_1, G20_{Q1}) = 0$, and $j(Q_1, G20_{Q1}) = j(P_1, G20_{Q1})$. Note that the presence of either of the Q_1-noncritical defeat-links is sufficient to give this result. So if we were to simplify the example by deleting S, that would not change the result.



Turning now to the hypergraph *HG*20, we have two separate instances of the defeater to represent in $HG20_{[\langle P1,Q1\rangle]}$. In the simple inference-graph $G20_{Q1}$, it is the maximum of the strengths of the two defeaters that is relevant to computing the effect of the defeaters on their mutual target. We can achieve the same result by having a single defeater in $HG20_{[\langle P1,Q1\rangle]}$ whose strength is the maximum of the strengths of the two instances, and that can be accomplished by having it supported by the two different arguments corresponding to the two instances. In that case, it gets the strength of the stronger argument. The two separate arguments for $(T\otimes\sim(Q_1\ \&\ Q_2))$ in $HG20_{[\langle P1,Q1\rangle]}$ are the $\langle P_1,Q_1\rangle$-independent argument from *S*, and the $\langle P_1,Q_1\rangle$-dependent argument from Q_2. The latter argument is based upon an argument for Q_2. That argument occurs in $HG20_{\langle P1,Q1\rangle}$ as well. Replicating it in $HG20_{[\langle P1,Q1\rangle]}$ involves some duplication of effort, but we will see in section 12.6 how that can be avoided.



12.5 Constructing HG_{λ} and $HG_{[\lambda]}$

To formulate precise rules for constructing HG_{λ} and $HG_{[\lambda]}$, we must consider the relationship between a defeat-link in HG and its instances in G. Inference-nodes in G correspond to arguments in HG. The identity of a defeat-link in G is determined by its root and its target, and the identity of the root is determined by its supporting argument. Recall that an argument is a kind of connected sub-tree of the inference-graph. We defined:

> An *argument in* an inference-hypergraph HG *for* a node φ is a minimal subset A of the nodes and support-links of the graph such that (1) if a node in A has any support-links in HG, exactly one of them is in A, (2) if a support-link is in A then the nodes in its support-link-basis are also in A, and (3) φ is in A.

An inference / defeat-path μ in HG has instances in G consisting of instances of the support-links and defeat-links constituting μ. A λ-circular-path μ in HG has an instance in G that passes

through a particular node N in G just in case it converges with a subpath in the argument for N and follows that subpath through N. This is diagrammed in figure 11. Here the λ-circular-path and the argument for N share the subpath $\langle L_3,L_2,L_1\rangle$. Note also that the link in the λ-circular-path immediately before the convergence must be a defeat-link. If it were a support-link, then it would pass through a different argument for the same basis element G of L_3, and hence would pass through a different inference-node in G (one supported by a different argument). Let us define:

Definition: In an inference-hypergraph, the δ-*neighborhood* of a λ-circular-path μ containing δ is that subpath $\langle L_1, \ldots ,L_n\rangle$ of μ such that (1) $\langle L_1, \ldots ,L_n,\delta\rangle$ is a subpath of μ, and (2) there is no support-link L^* such that $\langle L^*,L_1, \ldots ,L_n,\delta\rangle$ is a subpath of μ.

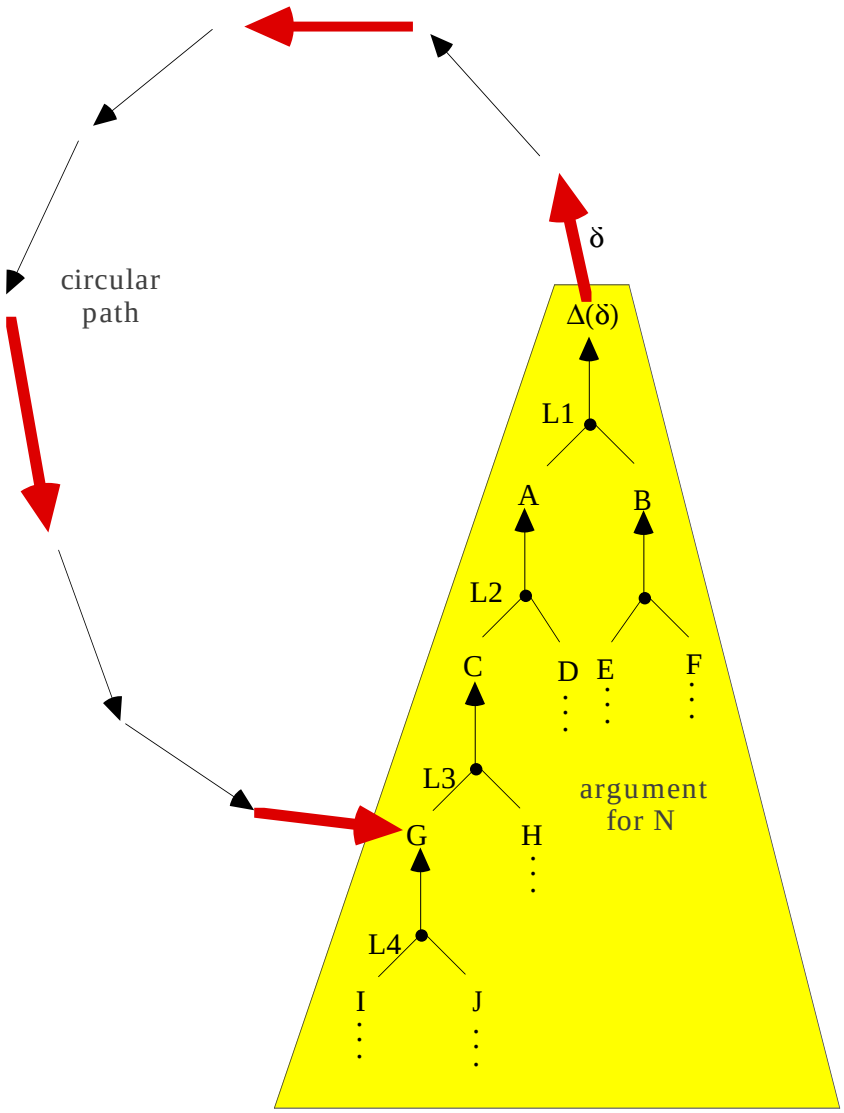


Figure 11. Convergence of Argument and Circular Path

In other words, the δ-neighborhood of a λ-circular-path μ consists of the subpath that results

from walking backwards from δ along support-links in μ until we either come to a defeat-link or come to the beginning of μ. In figure 11, the δ-neighborhood of the λ-circular-path is $\langle L_3,L_2,L_1\rangle$. Let us define:

Definition: If μ and ν are subpaths of an argument, μ is a *terminal subpath* of ν iff for some *n*, $\nu = \langle \nu_1,...,\nu_n\rangle$ and for some $i < n$, $\mu = \langle \nu_i,...,\nu_n\rangle$.

Definition: A λ-circular-path μ *converges with* an argument for δ in *HG* iff the δ-neighborhood of μ is a terminal subpath of a path through the argument, i.e., it is a subpath of the argument and Δ(δ) is the target of the last support-link in the neighborhood.

Then we have the simple theorem:

Theorem 52: If *N* is an inference-node in a hypergraph *HG* and μ is a λ-circular-path, a particular instance of *N* in the corresponding simple inference-graph *G* lies on an instance of μ in *G* iff μ converges with the corresponding argument for *N* in *HG*.

That is, a λ-circular-path in *HG* has an instance passing through a particular node *N* in *G* just in case the λ-circular-path converges with the argument for *N*. Note that there could be several different inference-nodes in *G* sharing the part of the argument that μ converges with, in which case μ represents multiple λ-circular-paths in *G*, and *N* lies on one of them.

Let δ be a λ-critical defeat-link in *HG*. An instance of δ in *G* is λ-noncritical iff it does not lie on a λ-circular-path in *G*. Equivalently, an instance of δ in *G* is noncritical iff no λ-circular-path in *HG* converges with the argument for the root of that instance of δ. Thus:

Theorem 53: A λ-critical defeat-link δ in *HG* has a λ-noncritical instance in *G* iff there is an argument for Δ(δ) in *HG* that does not converges with any λ-circular-path in *HG*.

Let us define:

Definition: An argument in *HG* is λ-*convergent* iff it converges with some λ-circular-path in *HG*.

So a λ-critical defeat-link δ in *HG* has a λ-noncritical instance in *G* iff there is a λ-nonconvergent argument for Δ(δ) in *HG*.

The values for the root Δ(δ) of a λ-critical defeat-link δ in HG_λ should be that produced by considering only λ-convergent arguments for Δ(δ). However, we do not incorporate complete arguments into HG_λ. In the simple inference-graph G_P, we make members of the node-basis of *P* and *P*-independent nodes initial. Analogously, in constructing HG_λ and $HG_{[\lambda]}$, we make members of the support-link-basis of λ and λ-independent nodes initial and assign them the values they have in *HG*. To make this precise we need the notion of an argument *from* a set of nodes. We defined an argument to be a tree structure that is anchored in initial nodes. We can similarly define:

Definition: An *argument for* a node φ *from* a set *X* of nodes in an inference-hypergraph *HG* is a minimal subset A of the nodes and support-links of the graph such that (1) if a node in A is not in *X* and it has any support-links in *HG*, exactly one of them is in A, (2) if a support-link is in A then the nodes in its support-link-basis are also in A, and (3) φ is in A.

The minimality condition requires that you cannot just leave out some of the nodes and links in A and still have an argument for *N* from *X*.

The arguments that make up HG_{λ} are the λ-convergent arguments for λ, ⊗λ, and ~λ, *from* members of the support-link-basis of λ and λ-independent nodes in *HG*. λ-critical defeat-links in HG_{λ} inherit their values from $HG_{[\lambda]}$ rather than having them computed in HG_{λ}. This corresponds to removing λ-critical defeat-links from the simple inference-graph. So whenever a support-link in one of the λ-convergent arguments in HG_{λ} has a λ-critical defeat-link δ, Δ(δ) is set aside to have its λ-noncritical value computed in $HG_{[\lambda]}$.

Turning to $HG_{[\lambda]}$, the value in $HG_{[\lambda]}$ for the root Δ(δ) of a λ-critical defeat-link δ in *HG* should be produced by considering all λ-nonconvergent arguments for Δ(δ) from λ-independent nodes in *HG*. The defeat-links whose roots should have values computed in $HG_{[\lambda]}$ are ⊗λ and ~λ, and all defeat-links set aside during the construction of HG_{λ}. So we just incorporate into $HG_{[\lambda]}$ all λ-nonconvergent arguments for these defeaters.

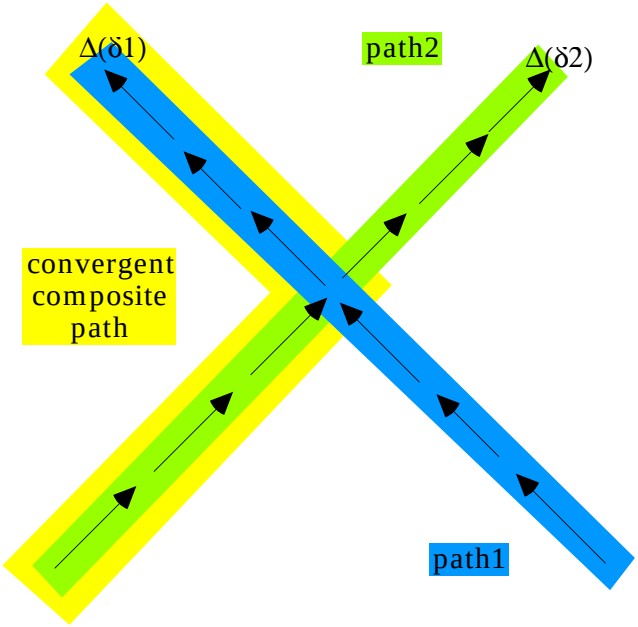


Figure 12. Convergent composite path.

The values computed in $HG_{[\lambda]}$ are supposed to be the λ-nonconvergent values of the roots of the defeat-links. It is clear that if we handled each Δ(δ) separately, in its own hypergraph, that is what we would be computing, because the hypergraph would just contain all of the λ-nonconvergent arguments for Δ(δ). But it might be supposed that by combining arguments for different defeat-link roots into a single hypergraph, we run the risk of introducing additional unintended arguments for Δ(δ) that converge with λ-circular-paths. No path in a λ-nonconvergent

argument can converge with a λ-circular-path, but paths taken from different arguments can cross. It might seem that this could create new hybrid paths that converge with a λ-circular-path and so should not be included in $\mathcal{HG}_{[\lambda]}$.. The situation is diagrammed in figure 12.

The intent in figure 12 is that $path_1$ to $\Delta(\delta_1)$ (blue or dark grey) and $path_2$ to $\Delta(\delta_2)$ (green or medium grey) do not converge with λ-circular-paths, but the hybrid path consisting of the first part of $path_2$ and the last part of $path_1$ (yellow or light grey) might converge with a λ-circular-path. However, if the hybrid path converges with a λ-critical path, then there is path from λ to the start of $path_2$. δ_1 and δ_2 are, by hypothesis, λ-critical defeat-links, so they lie on paths back to λ. Thus there is a path from $path_2$ to λ, and so if there is also a path from λ to $path_2$ then, contrary to hypothesis, $path_2$ must already converge with a λ-circular-path. Thus combining minimal λ-noncritical partial-arguments in a single hypergraph will not create new arguments lying on λ-circular-paths. Consequently, no λ-convergent arguments can be created by combining all the λ-nonconvergent arguments into the single hypergraph $\mathcal{HG}_{[\lambda]}$.

$\mathcal{HG}_{\lambda}$ can be constructed recursively. We want to compute the values of λ, ⊗λ and ~λ, so to construct $\mathcal{HG}_{\lambda}$, we begin by inserting λ together with the ⊗λ-neighborhoods and ~λ-neighborhoods of all λ-circular-paths in $\mathcal{HG}$. We add members of the support-link-basis of λ as initial nodes. These must then be expanded into arguments by working backwards from each node in a neighborhood, adding all its support-links, but stopping when we come to λ-independent nodes. For any support-link we add, we add members of its support-link-basis, and then we look at each of its defeat-links δ. If δ is not λ-critical, add it to $\mathcal{HG}_{\lambda}$ and add $\Delta(\delta)$ to $\mathcal{HG}_{\lambda}$, and then repeat the above steps. If δ is λ-critical, add it to $\mathcal{HG}_{\lambda}$, but put $\Delta(\delta)$ aside for addition to $\mathcal{HG}_{[\lambda]}$ instead. These λ-critical defeat-links will not have values computed in $\mathcal{HG}_{\lambda}$. Instead, they will inherit their values from the values (i.e., the maximum of the values of their λ-noncritical instances) of their roots in $\mathcal{HG}_{[\lambda]}$.

To this definition of $\mathcal{HG}_{\lambda}$ and $\mathcal{HG}_{[\lambda]}$, we must add the specification of the values of the initial elements in these hypergraphs:

If N is an initial node in $\mathcal{HG}_{\lambda}$ (such nodes are either members of the support-link-basis of λ or λ-independent), $j(N,\mathcal{HG}_{\lambda}) = j(N,\mathcal{HG})$.

If δ is in $\mathcal{HG}_{\lambda}$ and δ is a λ-critical defeat-link in $\mathcal{HG}$, $j(\delta,\mathcal{HG}_{\lambda}) = j(\Delta(\delta),\mathcal{HG}_{[\lambda]})$.

Finally, our evaluation rule becomes (where $\max(\varnothing) = 0$):

(DJH) Computing degrees of justification in an inference-hypergraph:

(1) If N is an inference-node in $\mathcal{HG}$, $j(N,\mathcal{HG}) = \max\{j(L,\mathcal{HG}) \mid L$ is a support-link for N in $\mathcal{HG}\}$.

(2) If δ is a defeat-link in $\mathcal{HG}$ and its root N is in $\mathcal{HG}$, $j(\delta,\mathcal{HG}) = j(N,\mathcal{HG})$.

(3) If λ is a support-link in $\mathcal{HG}$ and λ has no λ-dependent defeat-links, then if the basis of λ is $\{B_1,...,B_n\}$ and its reason-strength is ρ, $j(\lambda,\mathcal{HG}) = \min\{\rho,j(B_1,\mathcal{HG}),...,j(B_n,\mathcal{HG})\} \sim j(\blacktriangleleft\otimes\lambda\blacktriangleright,\mathcal{HG})$.

(4) If λ is a support-link in $\mathcal{HG}$ and λ has λ-dependent defeat-links, then $j(\lambda,\mathcal{HG}) = j(\lambda,\mathcal{HG}_{\lambda}) \sim \max\{j(\sim\lambda,\mathcal{HG}_{\lambda}), j(\otimes\lambda,\mathcal{HG}_{\lambda})\}$

Note that, as in the discussion of collaborative defeat, (4) can be rewritten equivalently as follows:

(4*) If λ is a support-link in HG with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ has λ-dependent defeat-links, then
$j(\lambda,HG) = \min\{\rho, j(B_1,HG),...,j(B_n,HG)\} \sim [j(\otimes\lambda,HG_{[\lambda]}) + \max\{j(\sim\lambda,HG_\lambda), j(\otimes\lambda,HG_\lambda)\}]$.

(DJH) is to be understood so that if any of $\otimes\lambda$, $\sim\lambda$, or $\blacktriangleleft\otimes\lambda\blacktriangleright$ are not present in the relevant inference-graphs then their contributions are omitted from the summation.

12.6 Exporting Support-Links

It was observed above that HG_λ and $HG_{[\lambda]}$ duplicate some of the same reasoning. For instance, in $HG20_{\langle P1,Q1\rangle}$ and $HG20_{[\langle P1,Q1\rangle]}$, the subargument from P_2 to Q_2 occurs in both inference-graphs. Furthermore, the computation of the degree of justification of Q_2 on the basis of that argument is identical for both inference-graphs. We could avoid such duplication of effort by just including that argument in HG_λ, and then importing its value into $HG_{[\lambda]}$ as needed. The crucial idea is that, just as defeat-links can be imported into HG_λ from $HG_{[\lambda]}$, so support-links could be exported from HG_λ to $HG_{[\lambda]}$. $HG_{[\lambda]}$ is supposed to contain the λ-nonconvergent arguments for various defeaters, but in fact it need not contain the arguments in their entirety. It need only contain the "upper" parts of the arguments, where it makes a difference whether we are considering λ-convergent or λ-nonconvergent arguments. Once we have extended all the paths in an argument beyond the point where they are determined to not converge with any δ-neighborhood, the rest of the argument will be the same regardless of which hypergraph it occurs in. So any support-link we come to after we have passed the point of possible convergence need only have its value computed in one of HG_λ or $HG_{[\lambda]}$. Some of these support-links may have values computed in HG_λ anyway, just in the course of computing the values for λ, $\otimes\lambda$, and $\sim\lambda$ in HG_λ, so we might as well compute them all there and export them to $HG_{[\lambda]}$ as needed.

To make this precise, we have to say exactly how much of a λ-nonconvergent argument must be included in $HG_{[\lambda]}$, before we relegate the rest to HG_λ. For this purpose, we need the notion of a *partial-argument*. A partial-argument for a node starts with the node and adds a support-link, then adds support-links for (some of) the members of the basis of the first support-link, and so on, but it need not go all the way to initial nodes. It can stop leaving some nodes "unsecured" by further support-links. A partial-argument represents the "top part" of a complete argument. We can define precisely:

Definition: A *partial-argument for N* is an argument A for N from some set X of inference-nodes.

A support-path through a partial-argument is understood as going from a support-link having an unsecured member of its basis to the conclusion of the argument. In other words, it traverses the entire argument. A *downward extension* of a support-path consists of a longer support-path that adds further support-links to the front of the first support-path. That is, the original path is a terminal subpath of the extension. Let us say that a λ-circular-path μ *converges with* a support-path for Δ(δ) iff the δ-neighborhood of μ is a terminal subpath of the support-path. Thus an argument

converges with μ iff some support-path through the argument does.

In deciding how much of a λ-nonvergent argument for Δ(δ) must be included in $HG_{[\lambda]}$, we follow support-paths ρ downwards from Δ(δ) until they are long enough that we know that no λ-circular-path can converge with any downward extension of ρ. Let us define:

Definition: A support-path ρ through a partial-argument is λ-*safe* iff for every λ-circular-path μ, μ does not converge with either ρ or a downward extension of ρ.

Definition: A partial-argument is λ-*safe* iff all of its support-paths are λ-safe.

Now we can give a precise characterization of $HG_{[\lambda]}$. In constructing HG_{λ}, we collect a set of defeaters Δ(δ) whose values are to be computed in $HG_{[\lambda]}$. This list of defeaters includes ⊗λ and ~λ. $HG_{[\lambda]}$ then consists of all minimal λ-safe partial-arguments for one of the Δ(δ)'s. These are partial-arguments that have been extended just far enough so that each path begins with a support-link that prevents it from converging with a λ-circular-path. $HG_{[\lambda]}$ will consist of all support-links in minimal λ-safe partial-arguments, together with the inference-nodes that are their targets. We do not automatically incorporate the basis elements of support-nodes. They will be incorporated automatically in support-links that do not begin paths, but the support-links that do begin paths will have unsecured bases. Some of these support-links will be λ-independent, in which case they simply inherit their value from *HG*. The remaining unsecured support-links will be marked as *unsecured in* $HG_{[\lambda]}$ and will be understood as inheriting their values from HG_{λ}. To make this work, we must add a clause to the recursive definition of HG_{λ} inserting these "exported" support-links into HG_{λ}. The other clauses of the recursive definition will then insert whatever additional support-links, inference-nodes, and defeat-links are required for the computation of the value of the exported support-links.

Stating this precisely, our recursive characterization of HG_{λ} and $HG_{[\lambda]}$ becomes:

Definition:

- Initialize HG_{λ} as containing λ, all support-links in ⊗λ-neighborhoods or ~λ-neighborhoods of λ-circular-paths in *HG*, and all of the targets of these support-links.
- Initialize $HG_{[\lambda]}$ as containing all support-links in minimal λ-safe partial-arguments for ⊗λ and ~λ, together with the inference-nodes that are their targets. Add any of these support-links whose bases are unsecured to HG_{λ}.
- Do the following repeatedly until nothing new is added to HG_{λ} or $HG_{[\lambda]}$:
  - If a support-link is in HG_{λ}, add the members of its basis to HG_{λ}.
  - If a node is in HG_{λ}, and it is not a member of the support-link-basis of λ and it is not λ-independent, add its support-links to HG_{λ}.
  - If a support-link γ is in HG_{λ} and δ is a λ-noncritical defeat-link for γ, add δ and Δ(δ) to HG_{λ}.
  - If a support-link γ is in HG_{λ} and δ is a λ-critical defeat-link for γ, if there are minimal λ-safe partial-arguments for Δ(δ) in *HG*, add δ to to HG_{λ} and insert all support-links contained in the minimal λ-safe partial-arguments into $HG_{[\lambda]}$ together with the inference-nodes that are their targets. Add any of these support-links whose bases are unsecured to HG_{λ} unless they are λ-independent.

To this definition of HG_{λ} and $HG_{[\lambda]}$, we must add the specification of the values of the initial elements in these hypergraphs:

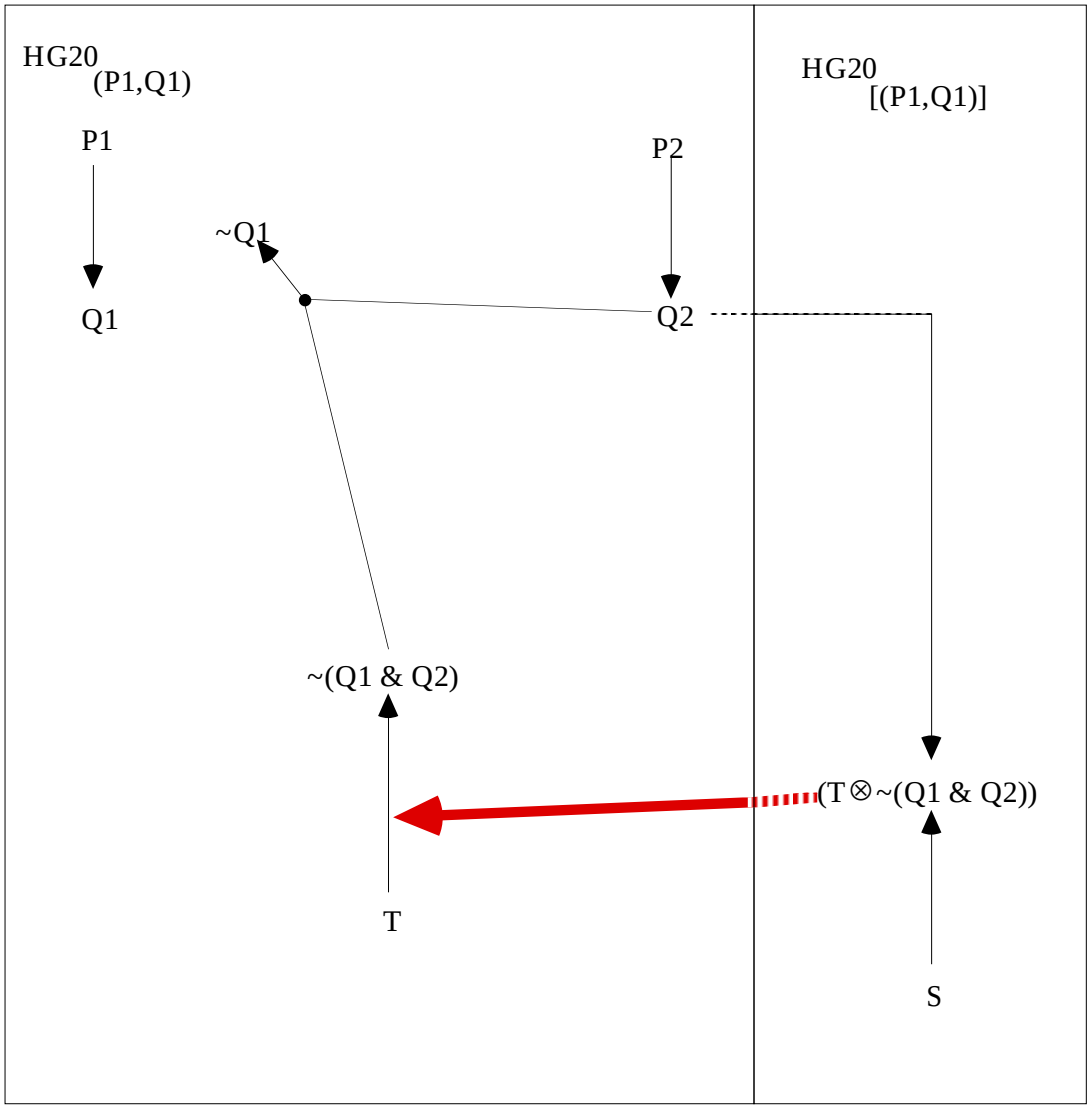
Definition:

If N is an initial node in HG_{λ} (such nodes are either members of the support-link-basis of λ or λ-independent), $j(N,HG_{\lambda}) = j(N,HG)$.

If δ is in HG_{λ} and δ is a λ-critical defeat-link in HG, $j(\delta,HG_{\lambda}) = j(\Delta(\delta),HG_{[\lambda]})$.

If L is a support-link in $HG_{[\lambda]}$ but some member of its basis is unsecured in $HG_{[\lambda]}$, $j(N,HG_{[\lambda]}) = j(N,HG_{\lambda})$.

The evaluation rule (DJH) remains unchanged.



To illustrate this construction, consider how it applies to *HG*20. In section 12.4 were led to construct $HG20_{\langle P1,Q1\rangle}$ and $HG20_{[\langle P1,Q1\rangle]}$ in such a way that both contain the subargument from P_2 to Q_2 to $(T\otimes\sim(Q_1 \,\&\, Q_2))$. But the present construction includes only the partial-argument from Q_2 to $(T\otimes\sim(Q_1 \,\&\, Q_2))$. The support-link $\langle Q_2,(T\otimes\sim(Q_1 \,\&\, Q_2))\rangle$ is the final unsecured link in the minimal λ-safe partial-argument for $(T\otimes\sim(Q_1 \,\&\, Q_2))$ that derives it from Q_2, so it is included in both $HG20_{\langle P1,Q1\rangle}$ and $HG20_{[\langle P1,Q1\rangle]}$ but only has a value computed in $HG20_{\langle P1,Q1\rangle}$. That value is then exported to $HG20_{[\langle P1,Q1\rangle]}$.

12.7 Minimal λ-Safe Partial-Arguments

In building $HG_{[\lambda]}$, we want to include all of the minimal λ-safe partial-arguments for Δ(δ). To implement this, we need an efficient algorithm for finding such partial-arguments and using them to construct $HG_{[\lambda]}$. We are aided in this by the following theorem:

Theorem 54: A support-path ρ for Δ(δ) is λ-safe iff for each δ-neighborhood η of a λ-circular-path μ, η is not a terminal subpath of ρ and ρ is not a terminal subpath of η.

Proof: If η is a terminal subpath of ρ then μ converges with ρ. If ρ is a terminal subpath of η, then η is a downward extension of ρ and η converges with η. So in either case, ρ is not λ-safe. Conversely, suppose ρ is not λ-safe. Then there is a λ-circular-path μ such that μ converges with either ρ or a downward extension of ρ. Let η be the δ-neighborhood of μ. Then either η is a terminal subpath of ρ or there is a downward extension ρ* of ρ such that η is a terminal subpath of ρ*. If there is a downward extension ρ* of ρ such that η is a terminal subpath of ρ*, then η is a terminal part of ρ*. If η is not a terminal subpath of ρ, then η must be a terminal part of ρ* that starts before the first member of ρ, so ρ is a terminal subpath of η. ■

So in collecting the minimal λ-safe support-paths, we just try all of the ways of extending support-paths downwards until they satisfy this condition.

We can do better, however, by appealing to λ-defeat-loops instead of λ-circular-paths. Recall that a λ-defeat-loop includes just the defeat-links from a λ-circular-path. We then have:

Theorem 55: If ρ is a support-path through a partial-argument to Δ(δ), and ρ_1 is the first member of ρ, then ρ is λ-safe iff for each λ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$, if for some *i*, $\delta = \delta_i$, then (1) if $i = 1$ then $\lambda\notin\rho$ and target(λ) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_1, and (2) if $i > 1$ then target(δ_{i-1})$\notin\rho$ and ultimate-target(δ_{i-1}) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_i.

Proof: Suppose $\langle\delta_1,...,\delta_k\rangle$ is a λ-defeat-loop and $\delta = \delta_1$. If $\lambda\in\rho$ then let ρ* be the terminal subpath of ρ that starts with λ. Then there is a λ-circular-path for which $\langle\delta_1,...,\delta_k\rangle$ is the corresponding λ-defeat-loop μ and which is such that ρ* is the δ-neighborhood of μ. So ρ is not λ-safe. If instead target(δ_i) is equal to or a node-ancestor of a member β of the support-link-basis of ρ_1, then there is a support-path ρ_0 from target(δ_i) to β. Then there is a λ-circular-path for which $\langle\delta_1,...,\delta_k\rangle$ is the corresponding λ-defeat-loop μ and which is such that the support-path $\rho_0{}^\wedge\rho$ that results from concatenating ρ_0 and ρ is the δ-neighborhood of μ. So ρ is not λ-safe.

Similarly, ρ is not λ-safe if $\delta = \delta_i$ where $i > 1$ and either target(δ_{i-1})$\in\rho$ and ultimate-target(δ_{i-1})

is equal to or a node-ancestor of a member of the support-link-basis of ρ_1.

Conversely, suppose ρ is not λ-safe and $\delta = \delta_i$ where $i > 1$. Then there is a λ-circular-path μ such that either the δ-neighborhood η of μ is a terminal subpath of ρ or ρ is a terminal subpath of η. If η is a terminal subpath of ρ then as target(δ_{i-1}) is the first member of η, target(δ_{i-1})$\in\rho$. If ρ is a terminal subpath of η then ultimate-target(δ_{i-1}) is equal to or a node-ancestor of a member of the support-link-basis of ρ_1. Similarly, if $\delta = \delta_1$ then either $\lambda\in\rho$ or target(λ) is equal to or a node-ancestor of a member of the support-link-basis of ρ_1. ■

We have characterized $\mathcal{HG}_{[\lambda]}$ in terms of λ-defeat-loops rather than λ-circular-paths, and that should make the computation more efficient. However, our recursive characterization of $\mathcal{HG}_\lambda$ begins by adding λ, all support-links in $\otimes\lambda$-neighborhoods or $\sim\lambda$-neighborhoods of λ-circular-paths in $\mathcal{HG}$ to $\mathcal{HG}_\lambda$. To avoid computing λ-circular-paths, we can reformulate this in terms of λ-defeat-loops. Define:

Definition: A support-link γ is *λ-critical for a defeat-link* δ iff (1) the target of γ is either $\Delta(\delta)$ or a node-ancestor of $\Delta(\delta)$, and (2) there is a λ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ such that for some i, $\delta = \delta_i$, and either (2a) $i = 1$ and either $\gamma = \lambda$ or target(λ) is a node-ancestor of some member of the basis of γ, or (2b) $i > 1$ and either target(δ_{i-1}) = γ or ultimate-target(δ_{i-1}) is a member of the basis or γ or a node-ancestor of some member of the basis of γ.

We have the following simple theorem:

Theorem 56: A support-link γ lies on the δ-neighborhood of some λ-circular-path iff γ is *λ-critical for* δ

Then our recursive characterization of $\mathcal{HG}_\lambda$ and $\mathcal{HG}_{[\lambda]}$ becomes:

Definition:

- Initialize $\mathcal{HG}_\lambda$ as containing λ, all support-links that are λ-critical for ◄$\otimes\lambda$► or ◄$\sim\lambda$►, and all of the targets of these support-links.
- Initialize $\mathcal{HG}_{[\lambda]}$ as containing all support-links in minimal λ-safe partial-arguments for $\otimes\lambda$ and $\sim\lambda$, together with the inference-nodes that are their targets. Add any of these support-links whose bases are unsecured to $\mathcal{HG}_\lambda$.
- Do the following repeatedly until nothing new is added to $\mathcal{HG}_\lambda$ or $\mathcal{HG}_{[\lambda]}$:
  - If a support-link is in $\mathcal{HG}_\lambda$, add the members of its basis to $\mathcal{HG}_\lambda$.
  - If a node is in $\mathcal{HG}_\lambda$, and it is not a member of the support-link-basis of λ and it is not λ-independent, add its support-links to $\mathcal{HG}_\lambda$.
  - If a support-link γ is in $\mathcal{HG}_\lambda$ and δ is a λ-noncritical defeat-link for γ, add δ and $\Delta(\delta)$ to $\mathcal{HG}_\lambda$.
  - If a support-link γ is in $\mathcal{HG}_\lambda$ and δ is a λ-critical defeat-link for γ, if there are minimal λ-safe partial-arguments for $\Delta(\delta)$ in $\mathcal{HG}$, add δ to to $\mathcal{HG}_\lambda$ and insert all support-links contained in the minimal λ-safe partial-arguments into $\mathcal{HG}_{[\lambda]}$ together with the inference-nodes that are their targets. Add any of these support-links whose bases are unsecured to $\mathcal{HG}_\lambda$ unless they are λ-independent.

Now let us turn to the algorithm for finding minimal λ-safe partial-arguments and using them to construct $\mathcal{HG}_{[\lambda]}$. Several observations are in order here. First, $\mathcal{HG}_{[\lambda]}$ is the union of all the minimal λ-safe partial-arguments for ⊗λ, ~λ, and whatever other Δ(δ)'s are needed for computing degrees of justification in $\mathcal{HG}_{\lambda}$. Note, however, that for building $\mathcal{HG}_{[\lambda]}$, we do not really need the λ-safe partial-arguments themselves. What we need is the set of all support-links occurring in them. So we want an algorithm that computes this set of support-links.

Let us define:

Definition: A path ρ through a partial-argument for Δ(δ) is λ-*convergent* iff there is a λ-circular-path μ whose δ-neighborhood is a terminal subpath of ρ.

A partial-argument is λ-convergent iff it contains a λ-convergent path. λ-convergent paths can also be characterized in terms of λ-defeat-loops:

Theorem 57: A path ρ through a partial-argument for Δ(δ) is λ-convergent iff there is a λ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ such that for some i, $\delta = \delta_i$, and either (1) $i = 1$ and $\lambda \in \rho$, and (2) $i > 1$ and $\text{target}(\delta_{i-1}) \in \rho$.

Theorems 55 and 57 provide useful characterizations of λ-safe paths and λ-convergent paths through partial-arguments. A simple algorithm for computing λ-safe partial-arguments starts with the empty path and then recursively extends paths downwards until they are either λ-convergent or λ-safe. The λ-convergent paths are rejected and the λ-safe partial-arguments are accepted. In constructing such paths, a useful observation is that if a path ρ is not λ-convergent and γ is a λ-independent support-link, then γ^ρ cannot be λ-convergent, because no λ-circular-path can include γ, so γ^ρ is λ-safe.

However, this is not a complete solution to the problem. It must be realized that a λ-safe path through a partial-argument need not be contained in a λ-safe partial-argument. For a partial-argument to be λ-safe, *all* of the paths through it must be λ-safe. An argument can contain λ-safe paths without this condition being satisfied. For example, if an argument supports a conjunction (*P* & *Q*) by containing subarguments for the conjuncts, *P* and *Q*, then there might be a λ-safe partial-argument for *P* but none for *Q*. In that case there is no λ-safe partial-argument for the conjunction.

To collect only the λ-safe paths that occur in λ-safe partial-arguments, we can modify the above algorithm. If a path ρ begins (at the bottom) with a support-link γ, we extend it downwards by adding a support-link ξ for some inference-node in the basis of γ. We reject paths that are λ-convergent, but we do not automatically accept those that are λ-safe. ξ will be a support-link for some member of the basis of γ. We only want to accept ξ^ρ (the result of adding ξ to the front of ρ) if there are λ-safe paths extending ρ and passing through *each* the members of the basis of γ. And this same condition must be imposed recursively on each member of each path.

To make this precise, we need three definitions. Where ρ is a support-path to Δ(δ) and β is a member of the basis of the first support-link in ρ (if ρ is nonempty), let us define:

Definition: A partial-argument for β is λ-safe *relative to* ρ iff for every path ρ_0 through the

partial-argument, ρ_0^ρ is λ-safe.

The idea is that if the partial-argument is λ-safe relative to β, then it represents some of the lower part of a partial-argument for $\Delta(\delta)$ that *might* be λ-safe. Our algorithm for finding minimal λ-safe arguments for $\Delta(\delta)$ will proceed by starting at $\Delta(\delta)$ and constructing longer and longer support-paths for it. When a path is λ-convergent, we reject it. When a path γ^ρ is λ-safe, we attempt to build a partial-argument for the target of its first support-link γ that is λ-safe relative to the rest of the path ρ. If we succeed then we attempt to expand that into a partial-argument for the target of the next support-link in ρ that is λ-safe relative to the rest of ρ. And so on until we obtain an argument that is λ-safe simpliciter. If at any point we cannot extend a λ-safe partial-argument upwards, then we dispose of it.

This construction can be done recursively. Define:

Definition:

Where ρ is a support-path to $\Delta(\delta)$ and β is a member of the basis of the first support-link in ρ (if ρ is nonempty), the λ-*subsidiary-support-links* for a node β relative to ρ is the set of all those support-links not already in ρ and lying on a minimal partial-argument for β that is λ-safe relative to ρ.

Where γ is a support-link for some member of the basis of the first support-link in ρ, the λ-*subsidiary-support-links* for support-link γ relative to ρ is the set of all those support-links not already in γ^ρ and lying on a minimal partial-argument for the target of γ that is λ-safe relative to ρ.

Clearly:

Theorem 58: A support-link occurs in a minimal λ-safe partial-argument for $\Delta(\delta)$ iff it is a λ-subsidiary support-link for $\Delta(\delta)$ relative to the empty path.

Next define:

Definition: A node is λ-*initial* iff it is either λ-independent or a member of the support-link-basis of λ.

When a previously non-λ-convergent path reaches a λ-initial node, it is guaranteed to be λ-safe. So the set of λ-subsidiary-support-links can be computed recursively by virtue of the following theorem, which also characterizes the unsecured links among them:

Theorem 59: If ρ is a support-path for $\Delta(\delta)$ not containing λ-safe or λ-convergent terminal subpaths:

(1) If ρ is the empty path or β is in the basis of the first member of ρ, then if β is λ-initial the set of λ-subsidiary-support-links for β relative to ρ is empty. If β is not λ-initial then the set of λ-subsidiary-support-links for β relative to ρ is the union of all the sets of λ-subsidiary-support-links for γ relative to ρ such that γ is a support-link for β, γ^ρ

is not λ-convergent, and γ∉ρ. One of these links is unsecured for β relative to ρ iff it is unsecured for one of the γ's relative to ρ

(2) If γ∉ρ, ρ is not λ-safe, and either γ is λ-independent or γ^ρ is λ-safe, the set of λ-subsidiary-support-links for γ relative to ρ is {γ}, and γ is unsecured unless it is λ-independent.

(3) If γ∉ρ and γ^ρ is not λ-safe and not λ-convergent:

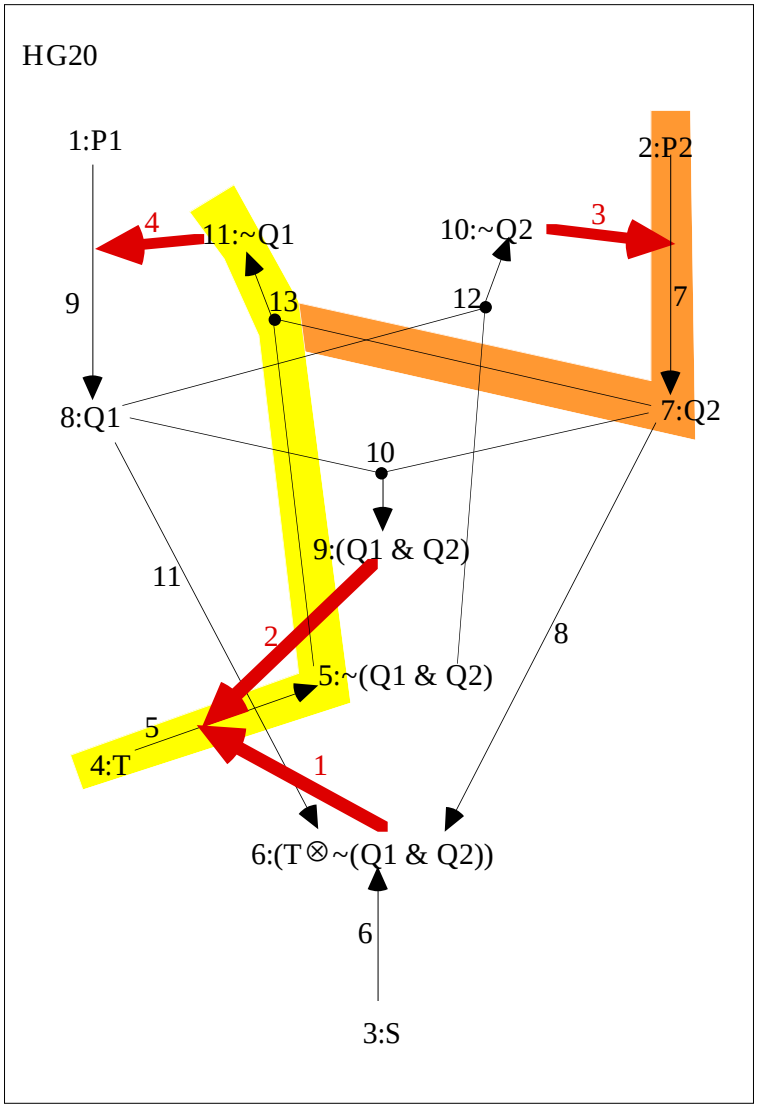
(a) if for some β in the basis of γ, β is not λ-initial and the set of λ-subsidiary-support-links for β relative to ρ is empty, the set of λ-subsidiary-support-links for γ relative to ρ is empty.

(b) otherwise, the set of λ-subsidiary-support-links for γ relative to ρ is the result of adding γ to the union of all the sets of λ-subsidiary-support-links relative to γ^ρ for members of the basis γ that are not λ-initial, and the unsecured links for γ relative to ρ are those links unsecured relative to γ^ρ for some member of the basis of γ that are not λ-initial.

(4) If γ∉ρ and γ^ρ is λ-convergent, the set of λ-subsidiary-support-links for γ relative to ρ is empty.

Theorem 59 constitutes a recursive characterization of λ-subsidiary-support-links and unsecured support-links and is readily implemented in LISP, with the result that the contents of $HG_{[\lambda]}$ consist of the λ-subsidiary-support-links and their targets for Δ(δ) relative to the empty path (for each Δ(δ) that is either ⊗λ, ~λ, or passed to $HG_{[\lambda]}$ from HG_{λ}). In applying clause (2), we can take advantage of the fact that if ρ is not λ-safe or λ-convergent and all members of the support-link-basis of γ are λ-initial then γ^ρ is λ-safe.

To illustrate the recursive computation provided by theorem 59, consider *HG*20 again. In computing the degree of justification of support-link #7, the hypergraph must be split. Having inserted support-link #9 in the critical side of the split, we must compute the minimal <support-link #7>-safe arguments for <defeat-link #4>. The following is a trace of the implementation (discussed in the appendix) of theorem 59:



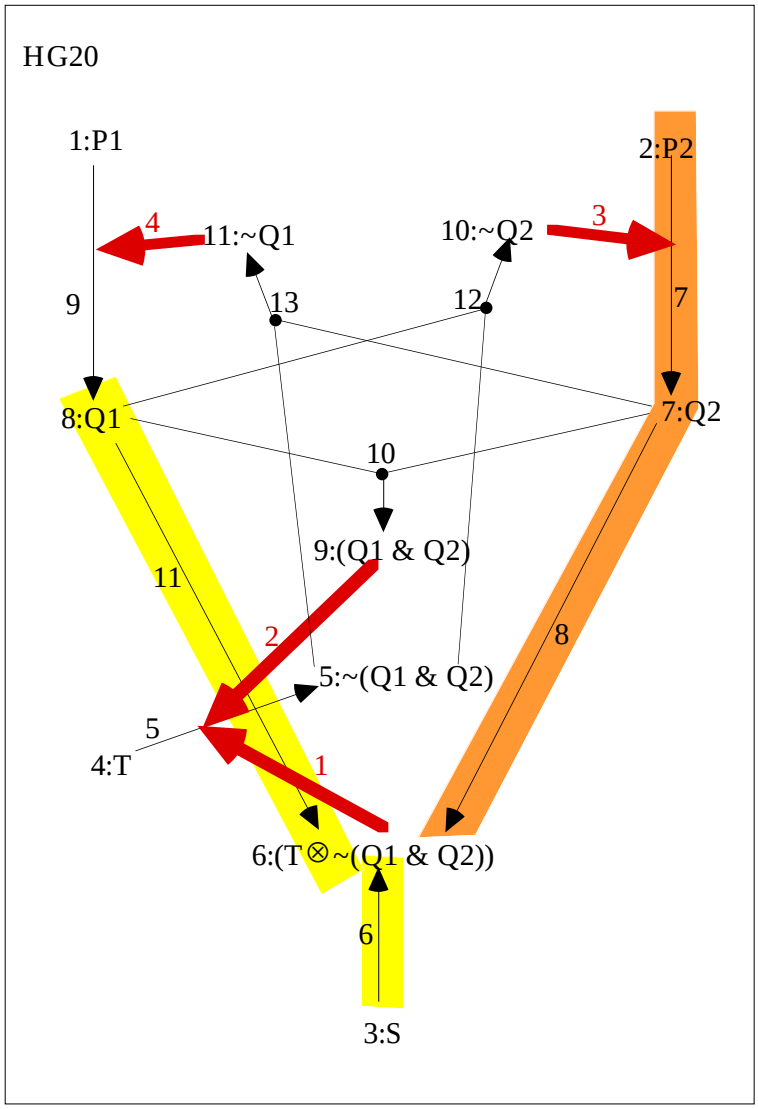
Computing minimal (<support-link #7>)-safe-arguments for <defeat-link #4>
.want subsidiary-support-links-for <Node 11> in (<support-link #7>) for <defeat-link #4> relative to path nil
..want subsidiary-support-links-for <support-link #13> in (<support-link #7>) for <defeat-link #4> relative to path nil
...want subsidiary-support-links-for <Node 5> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>)
....want subsidiary-support-links-for <support-link #5> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>)
....(<support-link #5><support-link #13>) is a (<support-link #7>)-safe support-path, so
....the set of subsidiary-support-links-for <support-link #5> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>) = (<support-link #5>)
...the set of subsidiary-support-links-for <Node 5> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>) = (<support-link #5>)
...want subsidiary-support-links-for <Node 7: Q2> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>)
....want subsidiary-support-links-for <support-link #7 for node 7> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>)
....(<support-link #7><support-link #13>) is (<support-link #7>)-convergent, so
....the set of subsidiary-support-links-for <support-link #7> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>) = nil
...the set of subsidiary-support-links-for <Node 7> in (<support-link #7>) for <defeat-link #4> relative to path (<support-link #13>) = nil
.the set of subsidiary-support-links-for <Node 11> in (<support-link #7>) for <defeat-link #4> relative to path nil = nil
The set of support-links in minimal (<support-link #7>)-safe-arguments for <defeat-link #4> = nil

The computation first finds the safe support-path ⟨<support-link #5>,<support-link #13>⟩, marked in yellow (or light grey) on the hypergraph. But to generate a safe argument, there must also be a safe support-path for the other member of the support-link-basis of support-link 13. The only possible support-path following that route is ⟨<support-link #7>,<support-link #13>⟩ (marked in orange, or darker grey), and that is found to be <support-link #7>-convergent, so the algorithm determines that there are no <support-link #7>-safe arguments for <defeat-link #4>.

Turning to <defeat-link #6>, the algorithm finds a minimal <support-link #7>-safe argument as follows:

Computing minimal (<support-link #7>)-safe-arguments for <defeat-link #1>
. want subsidiary-support-links-for <Node 6> in (<support-link #7>) for <defeat-link #1> relative to path nil
. . want subsidiary-support-links-for <support-link #11> in (<support-link #7>) for <defeat-link #1> relative to path nil
. . (<support-link #11>) is a (<support-link #7>)-safe support-path, so
. . the set of subsidiary-support-links-for <support-link #11> in (<support-link #7>) for <defeat-link #1> relative to path nil = (<support-link #11>)
. . want subsidiary-support-links-for <support-link #8 for node 6> in (<support-link #7>) for <defeat-link #1> relative to path nil
. . . want subsidiary-support-links-for <Node 7> in (<support-link #7>) for <defeat-link #1> relative to path (<support-link #8>)
. . . . want subsidiary-support-links-for <support-link #7 for node 7> in (<support-link #7>) for <defeat-link #1> relative to path (<support-link #8>)
. . . . (<support-link #7><support-link #8>) is (<support-link #7>)-convergent, so
. . . . the set of subsidiary-support-links-for <support-link #7 for node 7> in (<support-link #7>) for <defeat-link #1> relative to path (<support-link #8>) = nil
. . . the set of subsidiary-support-links-for <Node 7> in (<support-link #7>) for <defeat-link #1> relative to path (<support-link #8>) = nil
. . want subsidiary-support-links-for <support-link #6 for node 6> in (<support-link #7>) for <defeat-link #1> relative to path nil
. . <support-link #6> is (<support-link #7>)-independent, so
. . the set of subsidiary-support-links-for <support-link #6 for node 6> in (<support-link #7>) for <defeat-link #1> relative to path nil = (<support-link #6>)
. the set of subsidiary-support-links-for <Node 6> in (<support-link #7>) for <defeat-link #1> relative to path nil = (<support-link #6><support-link #11>)
The set of support-links in minimal (<support-link #7>)-safe-arguments for <defeat-link #1> = (<support-link #6><support-link #11>)

Here the algorithm finds two minimal <support-link #7>-safe arguments, one consisting of just <support-link #11> and the other consisting of just <support-link #6>. It also examines the support-path ⟨<support-link #7>,<support-link 8>⟩, but finds that it is <support-link #7>-convergent.



12.8 Refining the Computation

The constructions described above will begin by constructing the hypergraphs HG_{λ} and $HG_{[\lambda]}$. Evaluating a support-link μ in these hypergraphs may lead to the construction of further hypergraphs $(HG_{\lambda})_{\mu}$, $(HG_{\lambda})_{[\mu]}$, $(HG_{[\lambda]})_{\mu}$, and $(HG_{[\lambda]})_{[\mu]}$. And so on. Where $\langle\sigma_1,...,\sigma_n\rangle$ is a sequence of support-links λ and bracketed support-links [λ], let $HG_{\langle\sigma_1,...,\sigma_n\rangle} = (...(HG_{\sigma_n})_{\sigma_{n-1}} ...)_{\sigma_1}$. To refine the recursive computation and make it more readily implementable we can more or less repeat the analysis given for simple inference-graphs in section 11.2. Where σ is a sequence of support-links λ and bracketed support-links [λ], we want to give a recursive definition of $j_{\sigma}(x,HG)$ in such a way that $j_{\sigma}(x,HG) = j(x,HG_{\sigma})$. The recursive definition should be implementable without having to construct a sequence of inference-hypergraphs.

Where σ is a sequence of support-links λ and bracketed support-links [λ], let σ_1 be the first member of σ and let σ* be the rest of σ (its cdr). Where $\sigma = \langle\sigma_1,...,\sigma_n\rangle$, let $\lambda\text{^}\sigma = \langle\lambda,\sigma_1,...,\sigma_n\rangle$. Where *HG* is an inference-hypergraph, we define:

Definition: Where σ_1 is a support-link, a node θ is σ-*initial* iff θ has a value computed for σ and θ is either a member of the support-link-basis of σ_1 or θ is σ-independent.

Then we define, by simultaneous recursion:

Definition:

σ-*nodes*, σ-*links*, σ-*defeat-links*, are defined as follows:

- An inference-node is a ∅-node in *HG* iff it is in *HG*.
- A support-link is a ∅-link in *HG* iff it is in *HG*.
- A defeat-link is a ∅-defeat-link in *HG* iff it is in *HG*.
- Where σ is a sequence of support-links and bracketed support-links and λ is a support-link, let X and Y be the minimal sets satisfying the following conditions:
  - λ is in X, as are all support-links that are λ^σ-critical for ◄⊗λ► or ◄~λ►, and all of the targets of these support-links.
  - All support-links in λ^σ-safe partial-arguments for ⊗λ and ~λ are in Y, as are all of their targets. Any of these support-links that are σ-unsecured are also in X.
  - If a support-link is in X then all members of its basis are in X.
  - If a node is in X, and it is not λ^σ-initial, all of its σ-support-links are in X.
  - If a support-link γ is in X and δ is a σ*-defeat-link for γ that is not λ^σ-critical, δ and Δ(δ) are in X.
  - If a support-link γ is in X and δ is a σ*-defeat-link for γ that is λ^σ-critical, if there are minimal λ^σ-safe partial-arguments for Δ(δ) in *HG*, δ is in X and all support-links contained in the minimal λ^σ-safe partial-arguments are in Y, as are their targets. Any of these support-links that are σ-unsecured are also in X unless they are λ^σ-independent.
- An inference-node is a λ^σ-node iff it is in X.
- A support-link is a λ^σ-link iff it is in X.
- A defeat-link is a λ^σ-defeat-link iff either (1) it is in X, or (2) its root is in Y and it is λ^σ-critical.
- An inference-node is a [λ]^σ-node iff it is in Y.
- A support-link is a [λ]^σ-link iff it is in Y.
- A defeat-link is a [λ]^σ-defeat-link iff it is in Y.

Definition: A node θ is σ-*initial* iff σ_1 is a support-link and θ is either a member of the support-link-basis of σ_1 or θ is σ-independent.

Definition: A support-link γ is *contained in a minimal σ-safe argument* for Δ(δ) iff γ is a σ-subsidiary-support-link for Δ(δ) relative to the empty path.

Definition:

The σ-*subsidiary-support-links* and σ-*unsecured* support-links are defined recursively:

- If ρ is the empty path, or ρ is a list of support-links and β is in the basis of the first member of ρ, then if β is σ-initial the set of σ-subsidiary-support-links for β relative to ρ is empty. If β is not σ-initial then the set of σ-subsidiary-support-links for β relative to ρ is the union of all the sets of σ-subsidiary-support-links for γ relative to ρ such that γ is a support-link for β, γ^ρ is not σ-convergent, and $\gamma \notin \rho$. One of these links is σ-unsecured for β relative to ρ iff it is σ-unsecured for one of the γ's relative to ρ
- If $\gamma \notin \rho$, ρ is not σ-safe, and either γ is σ-independent or γ^ρ is σ-safe, the set of σ-subsidiary-support-links for γ relative to ρ is {γ}, and γ is σ-unsecured.
- If $\gamma \notin \rho$ and γ^ρ is not σ-safe and not σ-convergent:
  - if for some β in the basis of γ, β is not σ-initial and the set of σ-subsidiary-support-links for β relative to ρ is empty, the set of σ-subsidiary-support-links for γ relative to ρ is empty.
  - otherwise, the set of σ-subsidiary-support-links for γ relative to ρ is the result of adding γ to the union of all the sets of σ-subsidiary-support-links relative to γ^ρ for members of the basis γ that are not σ-initial, and the σ-unsecured links for γ relative to γ^ρ are those links σ-unsecured relative to ρ for some member of the basis of γ that are not σ-initial.
- If $\gamma \notin \rho$ and γ^ρ is σ-convergent then the set of σ-subsidiary-support-links for γ relative to ρ is empty.

Definition: A defeat-link δ of *HG* is σ-*critical* in *HG* iff (1) σ_1 is a support-link λ (not a bracketed support-link), (2) δ lies on a σ-defeat-loop μ in *HG*.

Definition: A defeat-link δ of *HG* is *hereditarily*-σ-*critical* in *HG* iff $\sigma \neq \varnothing$ and (1) δ is σ-*critical* in *HG* or (2) δ is hereditarily-σ*-*critical* in *HG*.

Definition: A sequence $\langle \delta_1,...,\delta_n \rangle$ of defeat-links is a λ-defeat-loop iff (1) the support-link-target of λ is a node-ancestor of the root of δ_1 but not of the root of any δ_k for $k > 1$, (2) λ is the target of δ_n, and (3) for each $k < n$, the ultimate-target of δ_k is equal to or an ancestor of the root of δ_{k+1}, but not of the root of δ_{k+j} for $j > 1$.

Definition: μ is a σ–*defeat-loop* in *HG* iff μ is defeat-loop in *HG* consisting entirely of σ*-defeat-links in *HG*.

Definition: μ is a σ-*support-path* in *HG* iff μ is a support-path in *HG* and μ consists entirely of σ-support-links in *HG*.

Definition: μ is a σ-*inference/defeat-path* in *HG* iff μ is an inference / defeat-path in *HG* and μ consists entirely of σ-support-links and σ-defeat-links in *HG*.

Definition: A defeat-link or support-link γ is σ-*independent* in *HG* iff $\sigma \neq \varnothing$ and there is no σ*-inference / defeat-path from σ_1 to γ in *HG*.

Definition: A node ψ is σ-*independent* in *HG* iff every σ-link for ψ is σ-independent in *HG*.

Definition: A σ-support-path ρ for Δ(δ) is σ-*convergent* iff for some support-link λ, $\sigma_1 = \lambda$, and there is a σ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ such that for some *i*, $\delta = \delta_i$, and (1) if $i = 1$ then $\lambda \in \rho$, and (2) if $i > 1$ then $\text{target}(\delta_{i-1}) \in \rho$.

Definition: A σ-support-path ρ for Δ(δ) is σ-*safe* iff for some support-link λ, $\sigma_1 = \lambda$, and where ρ_1 is the first member of ρ, for each σ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$, if for some *i*, $\delta = \delta_i$, then (1) if $i = 1$ then $\lambda \notin \rho$ and target(λ) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_1, and (2) if $i > 1$ then $\text{target}(\delta_{i-1}) \notin \rho$ and ultimate-target(δ_{i-1}) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_1.

Definition: A support-link γ is σ-*critical for* a defeat-link δ iff (1) for some support-link λ, $\sigma_1 = \lambda$, (2) the target of γ is either Δ(δ) or a node-ancestor of Δ(δ), and (3) there is a λ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ in σ* such that for some *i*, $\delta = \delta_i$, and (3a) if $i = 1$ then either $\gamma = \lambda$ or target(λ) is a node-ancestor of some member of the basis of γ, and (3b) if $i > 1$ then either $\text{target}(\delta_{i-1}) = \gamma$ or ultimate-target(δ_{i-1}) is a member of the basis or γ or a node-ancestor of some member of the basis of γ.

It is then trivial to prove by induction:

Theorem 60: If *HG* is an inference-hypergraph and σ is a finite sequence of support-links and bracketed support-links in *HG* then:

(1) a defeat-link *L* of *HG* is σ-critical in *HG* iff it is σ_1-critical in $HG_{\sigma*}$;
(2) μ is a σ–inference / defeat-path in *HG* iff μ is an inference / defeat-path in HG_σ;
(3) *L* is a σ-link in *HG* iff *L* is a support-link in HG_σ;
(4) *N* is a σ-node in *HG* iff *N* is an inference-node in HG_σ;
(5) δ is a σ-defeat-link in *HG* iff δ is a defeat-link in HG_σ;
(4) μ is a σ-support-path in *HG* iff μ is a support-path in HG_σ;
(5) a defeat-link or support-link γ is σ-independent of a support-link *L* in *HG* iff γ is independent of *L* in HG_σ;
(6) a support-path is λ^σ-convergent iff it is λ-convergent in HG_σ;
(7) a support-path is λ^σ-safe iff it is λ-safe in HG_σ;
(8) a node is σ-initial iff σ_1 is a support-link and the node is σ_1-initial in HG_σ;
(9) a support-link is a λ^σ-subsidiary support-link iff it is a λ-subsidiary support-link in HG_σ;
(10) a support-link is λ^σ-unsecured iff it is λ-unsecured in HG_σ.

Finally, we define recursively:

Definition: If *HG* is an inference-hypergraph:

(a) If ψ is initial in *HG* then $j_\sigma(\psi,HG) = j_0(\psi,HG)$;
(b) If ψ is an inference-node that is σ-independent in *HG* then $j_\sigma(\psi,HG) = j_{\sigma*}(\psi,HG)$;
(c) If ψ is an inference-node that is not σ-independent in *HG* then $j_\sigma(\psi,HG) = \max\{ j_\sigma(\lambda,HG) |$ λ is

a σ-link for ψ};

(d) If $\sigma_1 = [\lambda]$ and γ is a σ-unsecured support-link in HG, $j_\sigma(\gamma,HG) = j_{\lambda^\frown\sigma^*}(\gamma,HG)$;

(e) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ does not have a λ^σ-dependent defeater, then $j_\sigma(\lambda,HG) = \min\{\rho,j_\sigma(B_1,HG),...,j_\sigma(B_n,HG)\} \sim j_\sigma(\blacktriangleleft\otimes\lambda\blacktriangleright,HG)$;

(f) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ has a λ^σ-dependent defeater, then $j_\sigma(\lambda,HG) = \min\{\rho,j_\sigma(B_1,HG),...,j_\sigma(B_n,HG)\} \sim [j_{[\lambda]^\frown\sigma}(\otimes\lambda,HG) + \max\{j_{\lambda^\frown\sigma}(\sim\lambda,HG),j_{\lambda^\frown\sigma}(\otimes\lambda,HG)\}]$;

(g) If λ is a support-link and $\sigma_1 = \lambda$ and δ is a σ-critical defeat-link, then $j_\sigma(\delta,HG) = j_{[\lambda]^\frown\sigma^*}(\Delta(\delta),HG)$;

(h) If γ is a support-link and δ is a defeat-link that is not a σ-critical then $j_\sigma(\delta,HG) = j_\sigma(\Delta(\delta),HG)$.

The reason this constitutes a recursive definition is that when σ becomes long enough there will cease to be any σ-links.

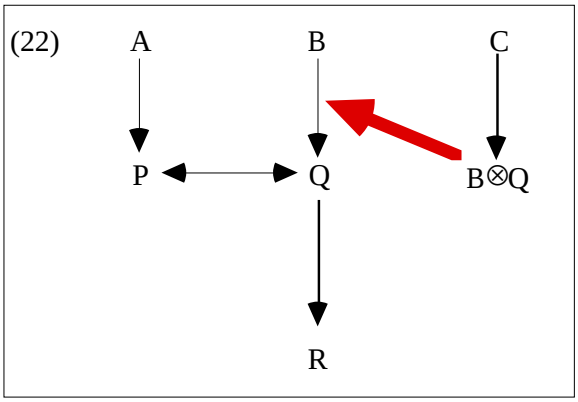
It is now trivial to prove by induction on the length of σ that:

Theorem 61: $j(\psi,HG_\sigma) = j_\sigma(\psi,HG)$.

Theorem 61 constitutes our desired recursive definition of $j_\sigma(\psi,HG)$. Because $j(\psi,HG) = j_\varnothing(\psi,HG)$, this also constitutes a recursive definition of $j(\psi,HG)$.

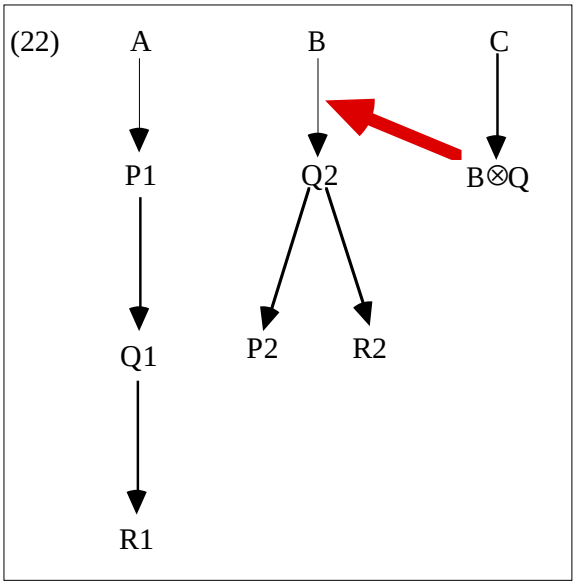
13. Circular Support-Paths

There is one case in which the computation of degrees of justification by (DJH) does not work. This occurs in an inference-hypergraph like (22) that contains an inference-loop. This can occur, for example, if P and Q are logically equivalent. Then either can be inferred from the other. In this hypergraph, there are two separate arguments to Q, viz., A → P → Q and B → Q. If one of these arguments is defeated, as it is in this hypergraph, the other one still supports Q.



If we apply the computation described by section 12 to $HG22$, it will not terminate. To compute $j(P,HG22)$ we would first have to compute $j(Q,HG22)$, and to compute $j(Q,HG22)$ we would first have to compute $j(P,HG22)$. The purpose of this section is to extend the analysis of

section 12 to handle inference-loops. To see how the computation should go, consider the equivalent simple inference-graph *G*22. Here we have two separate nodes for each of P, Q, and R, reflecting the fact that each is supported by two arguments. The arguments for Q2, P2, and R2 are defeated, but those for P1, Q1, and R1 are undefeated. So the degree of justification for R1 is that obtained from the argument $A \rightarrow P \rightarrow Q \rightarrow R$.



The degree of justification for a conclusion should be the maximum of that conveyed by the different arguments for that conclusion. In a simple inference-graph, different arguments are represented by separate nodes, but in an hypergraph different arguments are subtrees terminating on a common node. For the purpose of this paper, dependence was defined in terms if inference / defeat-paths. *Logical* dependence is defined in terms of inference-paths, ignoring defeat-links. More precisely, a node φ is logically dependent on a node θ iff θ is a node-ancestor of φ. Inference loops arise when a node is logically dependent on itself. In that case, there is some inference-path from the node φ to itself. This inference-path cannot be part of an argument, because arguments are required to be noncircular. The recursive computation of the degree of justification of a node must thus avoid going down inference-paths containing that node. There is no such restriction on computing the degree of justification for a defeater for a support-link for the node. The bulk of the paper has dealt with how the computation goes when a support-link has a self-dependent defeater. But in computing the degree of justification of a support-link, we must do two things. We must compute the degrees of justification of the elements of its basis, and we must compute the degrees of justification of its defeaters. We now know how to do the latter, but in computing the degrees of justification of the elements of the basis we must avoid going down circular inference-paths.

This can be handled by keeping track of for what nodes the recursive computation is attempting to compute degrees of justification. In computing the degree of justification for φ, we look at each support-link λ for φ. If the elements of the support-link-basis of λ are logically independent of φ, then we simply compute the degree of justification for λ. But if some element β of the support-link-basis of λ is logically dependent on φ, then we must compute a degree of justification for it *independent of* φ. To do this we walk down the various inference-paths leading to β, keeping

track of the fact that we want a β-independent degree of justification. If a path reaches a node θ that is logically independent of β, then the recursion returns degree of justification of θ and uses that in computing the next value as the recursion unwinds. If a path reaches the node β itself, it returns 0 and uses that in computing the next value as the recursion unwinds. If that path encounts a node that is logically dependent on β, the computation keeps going, extending the path downwards until it either reaches β or a node logically independent of β. Because an inference-node can have multiple support-links, some ways of extending the path downwards may terminate on β, and others may terminate on nodes that are logically independent of β. To illustrate, the course of computation for *HG*22 is diagrammed in figure 13.

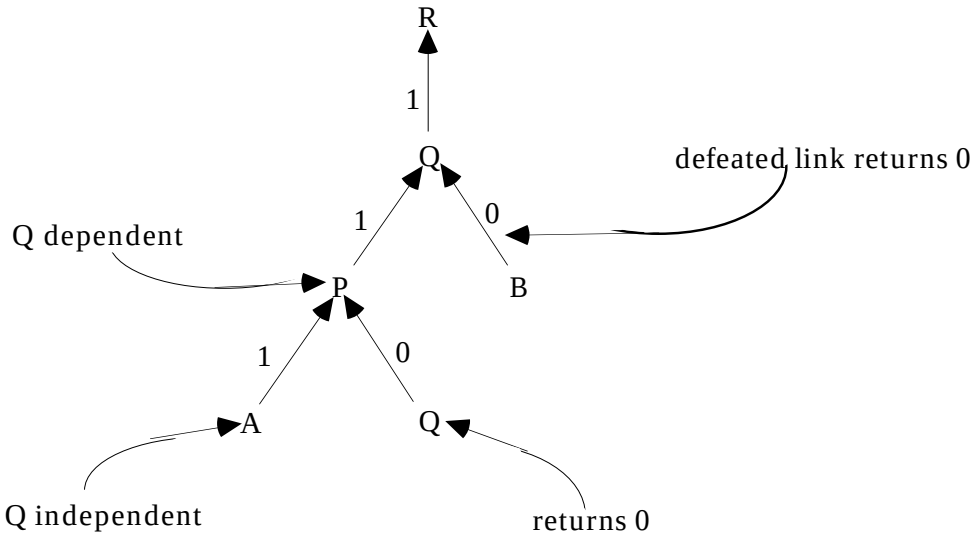


Figure 13. Computing Q-independent values

Formally, we accommodate this by adding a third variable to j_σ, writing $j_\sigma(x,HG,D)$, where D is the set of nodes the computed value should be logically independent of. D can contain more than a single node, because in computing the φ-independent value of φ we may encounter another node θ that is logically dependent on itself. We then revise the recursive definition of j_σ as follows:

Definition: If *HG* is an inference-hypergraph:

(a) If ψ is initial in *HG* then $j_\sigma(\psi,HG,D) = j_0(\psi,HG,\varnothing)$;

(b) If $\psi \in D$ then $j_\sigma(\psi,HG,D) = 0$;

(c) If $\theta \in D$ and θ is not a node-ancestor of ψ then where D^* results from removing θ from D, $j_\sigma(\psi,HG,D) = j_\sigma(\psi,HG,D^*)$.

(d) If ψ is an inference-node that is σ-independent in *HG* then $j_\sigma(\psi,HG,D) = j_{\sigma^*}(\psi,HG,D)$;

(e) If ψ is an inference-node that is not σ-independent in *HG* then $j_\sigma(\psi,HG,D) = \max\{ j_\sigma(\lambda,HG,\psi^\wedge D) | \lambda$ is a σ-link for $\psi\}$;

(f) If $\sigma_1 = [\lambda]$ and γ is a σ-unsecured support-link in *HG*, $j_\sigma(\gamma,HG,D) = j_{\lambda^\wedge\sigma^*}(\gamma,HG,D)$;

(g) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ,

and λ does not have a λ^σ-dependent defeater, then $j_\sigma(\lambda,\mathit{HG},D)$ = $\min\{\rho,j_\sigma(B_1,\mathit{HG},D),...,j_\sigma(B_n,\mathit{HG},D)\} \sim j_\sigma(\blacktriangleleft\otimes\lambda\blacktriangleright,\mathit{HG},\varnothing)$;

(h) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ has a λ^σ-dependent defeater, then $j_\sigma(\lambda,\mathit{HG})$ = $\min\{\rho,j_\sigma(B_1,\mathit{HG},D),...,j_\sigma(B_n,\mathit{HG},D)\} \sim [j_{[\lambda]\urcorner\sigma}(\otimes\lambda,\mathit{HG},\varnothing)$ + $\max\{j_{\lambda\urcorner\sigma}(\sim\lambda,\mathit{HG},\varnothing),j_{\lambda\urcorner\sigma}(\otimes\lambda,\mathit{HG},\varnothing)\}]$;

(i) If λ is a support-link and $\sigma_1 = \lambda$ and δ is a σ-critical defeat-link, then $j_\sigma(\delta,\mathit{HG},\varnothing)$ = $j_{[\lambda]^\sigma*}(\Delta(\delta),\mathit{HG},\varnothing)$;

(j) If γ is a support-link and δ is a defeat-link that is not a σ-critical then $j_\sigma(\delta,\mathit{HG},\varnothing)$ = $j_\sigma(\Delta(\delta),\mathit{HG},\varnothing)$.

14. Conclusions

The topic of degrees of justification resulting from defeasible reasoning is virtually unexplored in AI. There is a massive literature on degrees of probability (and the related Dempster-Shafer theory), but this paper partly argues and partly assumes (referring to arguments given elsewhere) that degrees of justification are not probabilities, in the sense that they do not conform to the probability calculus.

The starting point of the present theory is the observation that defeaters that are too weak to defeat an inference may nevertheless diminish the degree of justification of its conclusion. The search for a semantics for defeasible reasoning that is compatible with diminishing leads to a new way of computing degrees of justification recursively. A consequence of this analysis is the principle of collaborative defeat, wherein a pair of defeaters can defeat an inference when they are individually too weak to do that. Work is currently underway to implement this new computation of degrees of justification in OSCAR.

References

Barnard, G. A.

1949 "Statistical inference", *Journal of the Royal Statistical Society B, II*, 115-149.

1966 "The use of the likelihood function in statistical practice", *Proceedings v Berkeley Symposium on Mathematical Statistics and Probability I*, 27-40.

Birnbaum, Allan

1962 "On the foundations of statistical inference", *Journal of the American Statistical Association 57*, 269-326.

Chesñevar, Carlos, Ana Babriela Maguitman, and Ronald Loui

2000 "Logical models of argument", *ACM Computing Surveys* **32**, 337-383.

Covington, Michael, Donald Nute, and Andre Vellino

1997 *Prolog Programming in Depth*. Second edition. Prentice-Hall, Englewood Cliffs, NJ.

Dempster, A. P.

1968 "A generalization of Bayesian inference", *Journal of the Royal Statistical Society, Series B* **30**, 205-247.

Dung, P. M.

1995 "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and *n*-person games", *Artificial Intelligence* **77**, 321-357.

Edwards, A. W. F.

1972 *Likelihood*. Cambridge: Cambridge University Press.

Fisher, R. A.

1922 "On the mathematical foundations of theoretical statistics", *Philosophical Transactions of the Royal Society A*, *222*, 309-368.

Hacking, Ian

1965 *Logic of Statistical Inference*. Cambridge: Cambridge University Press.

Hanks, Steve, and McDermott, Drew

1986 "Default reasoning, nonmonotonic logics, and the frame problem", AAAI-86.

1987 "Nonmonotonic logic and temporal projection", *Artificial Intelligence* **33**, 379-412.

Kyburg, Henry

1961 *Probability and the Logic of Rational Belief*. Middletown, Conn.: Wesleyan University Press.

1974 *The Logical Foundations of Statistical Inference*. Dordrecht: Reidel.

Levi, Isaac

1977 "Direct inference". *Journal of Philosophy* **74**, 5-29.

Makinson, D. and Schlechta, K.

1991 "Floating conclusions and zombie paths: Two deep difficulties in the 'directly skeptical' approach to inheritance nets", *Artificial Intelligence* **48**, 199-209.

McCarthy, John

1986 "Applications of circumscription to formalizing common sense knowledge." *Artificial Intelligence* **26**, 89-116.

McDermott, Drew

1982 "A temporal logic for reasoning about processes and plans", *Cognitive Science* **6**, 101-155.

Nute, Don

1992 "Basic defeasible logic." In L. Farias del Cerro and M. Penttonen (eds.), *Intensional Logics for Programming*, Oxford University Press, 125-154.

1999 "Norms, priorities, and defeasibility." In P. McNamara and H. Prakken (eds.), *Norms, Logics and Information Systems*, IOS Press, Amsterdam, 201-218.

Peal, Judea

1988 *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan Kaufmann.

Pollock, John

1970 "The structure of epistemic justification", *American Philosophical Quarterly*, monograph series 4: 62-78.

1974 *Knowledge and Justification*, Princeton University Press.

1983 "Epistemology and probability", *Synthese* **55**, 231-252.

1987 *Contemporary Theories of Knowledge*, Rowman and Littlefield.

1990 *Nomic Probability and the Foundations of Induction*, Oxford University Press.
1994 "Justification and defeat", *Artificial Intelligence* **67**, 377-408.
1995 *Cognitive Carpentry*, MIT Press.
1997 "Reasoning about change and persistence: a solution to the frame problem", *Nous* **31**, 143-169.
1998 "Perceiving and reasoning about a changing world", *Computational Intelligence* **14**, 498-562.
1998a "Degrees of Justification", in P. Weingartner, G. Schurz and G. Dorn (Eds.), *The Role of Pragmatics in Contemporary Philosophy. (Proceedings of the 20th International Wittgenstein Symposium 1997 Kirchberg/Wechsel, Austria)*, Hoelder-Pichler Tempsky publishers, Vienna, 207-223.
1998b "The logical foundations of goal-regression planning in autonomous agents", *Artificial Intelligence* **106**, 267-335.
2001 "Defeasible reasoning with variable degrees of justification", *Artificial Intelligence* **133** (2001), 233-282.

Pollock, John, and Joe Cruz
1999 *Contemporary Theories of Knowledge*, 2nd edition. Rowman and Littlefield.

Poole, David
1988 "A logical framework for default reasoning", *Artificial Intelligence* **36**, 27-47.

Prakken, H. and G.A.W. Vreeswijk
2002 "Logics for Defeasible Argumentation", to appear in *Handbook of Philosophical Logic*, 2nd Edition, vol. 5, ed. D. Gabbay and F. Guenthner, Kluwer: Dordrecth.

Reichenbach, Hans
1949 *A Theory of Probability*. Berkeley: University of California Press. (Original German edition 1935).

Reiter, Raymond
1980 "A logic for default reasoning", *Artificial Intelligence* **13**, 81–132.

Sandewall, Erik
1972 "An approach to the frame problem and its implementation". In B. Metzer & D. Michie (eds.), *Machine Intelligence* **7**. Edinburgh: Edinburgh University Press.

Shafer, G.
1976 *A Mathematical Theory of Evidence*. Princeton: Princeton University Press.

Simari, G. R., and Loui, R. P.
1992 "A mathematical treatment of defeasible reasoning and its implementation", *Artificial Intelligence* **53**, 125–158.

Touretzky, David
1984 "Implicit orderings of defaults in inheritance systems", *Proceedings of AAAI-84*.

Touretzky, David, John Horty, and Richmond Thomason
1987 "A clash of intuitions: the current state of nonmonotonic multiple inheritance systems", *IJCAI87*, 476-482.

Verheij, Bart
1996 *Rules, Reasons, Arguments*. PhD dissertation, University of Maastricht, The Netherlands.

Appendix: LISP Code

The paper formulates an algorithm for computing degrees of justification in hypergraphs. The algorithm begins by constructing the hypergraphs HG_{λ} and $HG_{[\lambda]}$. Evaluating a support-link μ in these hypergraphs may lead to the construction of further hypergraphs $(HG_{\lambda})_{\mu}$, $(HG_{\lambda})_{[\mu]}$, $(HG_{[\lambda]})_{\mu}$, and $(HG_{[\lambda]})_{[\mu]}$. And so on. Where $\langle\sigma_1,...,\sigma_n\rangle$ is a sequence of support-links λ and bracketed support-links $[\lambda]$, let $HG_{\langle\sigma_1,...,\sigma_n\rangle} = (...(HG_{\sigma_n})_{\sigma_{n-1}} ...)_{\sigma_1}$. Where σ is a sequence of support-links λ and bracketed support-links $[\lambda]$, the paper gives a recursive definition of $j_{\sigma}(x,HG)$ having the consequence that $j_{\sigma}(x,HG) = j(x,HG_{\sigma})$. This appendix describes the implementation of the algorithm in OSCAR.

We begin with OSCAR_3.33, and add the following fields to inference-nodes, support-links, and defeat-links:

```
(defstruct (inference-node
   ...
    (node-justifications NIL)  ;; list of pairs (sigma.val) used by justification
    (node-in (list NIL))  ;; list of  lists of links
    (node-dependencies NIL)   ;; list of sigmas
    )

(defstruct (support-link
   ...
    (unsecured-link? NIL) ;; list of sigmas
    (link-defeat-loops T) ;; defeat-loops from link to link
    (link-justifications NIL)  ;; list of pairs (sigma.val) used by justification
    (link-in (list NIL))  ;; list of sigmas
    (link-dependencies NIL)  ;; list of sigmas
    )

(defstruct (defeat-link
   ...
   (critical-link? NIL)  ;; list of (sigma.t) or (sigma.NIL)
   (defeat-link-justifications NIL)  ;; list of pairs (sigma.val).
   (defeat-link-in (list NIL))  ;; list of  lists of links
   )
```

The computation of degrees of justification is done by COMPUTE-DEGREES-OF-JUSTIFICATION, which is called by UPDATE-BELIEFS. This begins by nulling the remembered values of previously computed parameters for all newly derived support-links, and all support-links, defeat-links, and inference-nodes affected by them. Then it recomputes the degrees of justification of all the affected items.

```
(defunction compute-degrees-of-justification ()
   (dolist (link *new-links*) (reset-memories link))
   (dolist (link *new-links*) (compute-affected-justifications link)))

(defunction reset-memories (link)
   (setf (link-justifications link) NIL)
   (setf (link-defeat-loops link) T)
   (setf (unsecured-link? link) NIL)
   (setf (link-in link) (list NIL))
```

```
  (setf (link-dependencies link) NIL)
  (let ((node (support-link-target link)))
    (setf (node-justifications node) NIL)
    (setf (node-in node) (list NIL))
    (setf (node-dependencies node) NIL)
    (dolist (L (consequent-links node))
      (when (link-justifications L) (reset-memories L)))
    (dolist (dl (supported-defeat-links node))
      (setf (defeat-link-justifications dl) NIL)
      (setf (critical-link? dl) NIL)
      (setf (defeat-link-in dl) (list NIL))
      (let ((target (defeat-link-target dl)))
      (when (link-justifications target) (reset-memories target))))))

(defunction compute-affected-justifications (link)
  (let ((node (support-link-target link)))
      (unless (assoc NIL (node-justifications node))
       (compute-node-justification node link N)
       (dolist (L (consequent-links node))
          (unless (assoc NIL (link-justifications L))
            (compute-affected-justifications L)))
       (dolist (DL (supported-defeat-links node))
          (let ((L (defeat-link-target DL)))
            (unless (assoc NIL (link-justifications L))
              (compute-affected-justifications L)))))))
```

Where σ is a sequence of support-links λ and bracketed support-links [λ] (represented in the implementation by unit sets of support-links), let σ_1 be the first member of σ and let σ* be the rest of σ (its cdr). Where $\sigma = \langle\sigma_1,...,\sigma_n\rangle$, let $\lambda\hat{}\sigma = \langle\lambda,\sigma_1,...,\sigma_n\rangle$ (i.e., the result of consing λ into σ). The computation of degrees of justification is then performed in accordance with the following definition:

Definition: If *HG* is an inference-hypergraph:

(a) If ψ is initial in *HG* then $j_\sigma(\psi,HG) = j_0(\psi,HG)$;

(b) If ψ is an inference-node that is σ-independent in *HG* then $j_\sigma(\psi,HG) = j_{\sigma*}(\psi,HG)$;

(c) If ψ is an inference-node that is not σ-independent in *HG* then $j_\sigma(\psi,HG) = \max\{ j_\sigma(\lambda,HG) |$ λ is a σ-link for ψ$\}$;

(d) If $\sigma_1 = [\lambda]$ and γ is a σ-unsecured support-link in *HG*, $j_\sigma(\gamma,HG) = j_{\lambda\hat{}\sigma*}(\gamma,HG)$;

(e) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ does not have a λ^σ-dependent defeater, then $j_\sigma(\lambda,HG) = \min\{\rho,j_\sigma(B_1,HG),...,j_\sigma(B_n,HG)\} \sim j_\sigma(\blacktriangleleft\otimes\lambda\blacktriangleright,HG)$;

(f) If λ is a support-link that is not σ-unsecured, with basis $\{B_1,...,B_n\}$ and reason-strength ρ, and λ has a λ^σ-dependent defeater, then $j_\sigma(\lambda,HG) = \min\{\rho,j_\sigma(B_1,HG),...,j_\sigma(B_n,HG)\} \sim [j_{[\lambda]\hat{}\sigma}(\otimes\lambda,HG) + \max\{j_{\lambda\hat{}\sigma}(\sim\lambda,HG),j_{\lambda\hat{}\sigma}(\otimes\lambda,HG)\}]$;

(g) If λ is a support-link and $\sigma_1 = \lambda$ and δ is a σ-critical defeat-link, then $j_\sigma(\delta,HG) = j_{[\lambda]\hat{}\sigma*}(\Delta(\delta),HG)$;

(h) If γ is a support-link and δ is a defeat-link that is not a σ-critical then $j_\sigma(\delta,HG) = j_\sigma(\Delta(\delta),HG)$.

Here we are employing the following definitions:

Definition: An *inference/defeat-path* from a support-link γ to a support-link or defeat-link λ in an

inference-hypergraph *HG* is a sequence of support-links and defeat-links in *HG* such that (1) the first link is γ; (2) the last link in the path is λ; (3) the root of each defeat-link after the first member of the path is the target of the preceding link; (4) some member of the basis of of each support-link after the first member of the path is the ultimate-target of the preceding link; and (5) the path does not contain an internal loop, i.e., no two links in the path have the same tultimate-arget.

Definition: μ is a σ-*inference/defeat-path* in *HG* iff μ is an inference/defeat-path in *HG* and μ consists entirely of σ-support-links and σ-defeat-links in *HG*.

Definition: A defeat-link or support-link γ is σ-*independent* in *HG* iff $\sigma \neq \varnothing$ and there is no σ*-inference/defeat-path from σ_1 to γ in *HG*.

Definition: A node ψ is σ-*independent* in *HG* iff every σ-link for ψ is σ-independent in *HG*.

However, in the implementation we want to avoid having to construct inference/defeat-paths, using defeat-loops in their place. This means that we cannot directly implement this definition of independence. Instead, we can make use of the fact that the recursive computation of degrees of justification traverses backwards along inference/defeat-paths. Thus we should be able to detect dependence at the same time we compute degrees of justification. Our principal interest is in detecting λ-dependent defeat-links for a support-link λ. To accomplish this, let us keep track of the sequences of support-links (paths) that the recursion traverses in attempting to compute the degrees of justification of defeaters for λ relative to σ (i.e., in HG_σ). If a path starting from λ and passing through a defeater for λ eventually encounters λ again, then it follows that λ has a λ^σ-dependent defeater. More generally, if a path from a defeater for λ contains a support-link γ and then continues on to encounter γ again, it follows that γ is γ^σ-dependent. If there is no path continuing on from a defeater for γ and subsequently encountering λ, then γ has no λ^σ-dependent defeater, so its value can be computed in HG_σ without splitting the latter into $HG_{\lambda\wedge\sigma}$ and $HG_{[\lambda]\wedge\sigma}$. However, to compute the degree of justification of γ in HG_σ, we will have to split HG_σ into $HG_{\gamma\wedge\sigma}$ and $HG_{[\gamma]\wedge\sigma}$. On the other hand, if there is a path continuing on from γ and subsequently encountering γ, then γ has a λ^σ-dependent defeater, so to compute its value in HG_σ we must first compute its values in $HG_{\lambda\wedge\sigma}$ and $HG_{[\lambda]\wedge\sigma}$. If γ remains γ-dependent in one of these hypergraphs, then we will have to split the hypergraphs further, e.g., into $HG_{\gamma\wedge\lambda\wedge\sigma}$ and $HG_{[\gamma]\wedge\lambda\wedge\sigma}$ and into $HG_{\gamma\wedge[\lambda]\wedge\sigma}$ and $HG_{[\gamma]\wedge[\lambda]\wedge\sigma}$. However, for computing the value of λ in HG_σ, we have no reason to compute the value of γ in HG_σ.That value will not play a role in computing the value of λ in HG_σ. In general, when we have several support-links in the path for which there are extensions of the path leading back to these same support-links, they are all self-dependent in HG_σ, but the only one whose value we are interested in is the first one in the path.

To keep track of all this, we construct paths by adding a support-link to the path whenever the course of computation leads to a defeater for that support-link. On a path on which the computation is able to compute a numerical value for the degree of justification of each item, we let the recursion unwind normally. But when the recursion encounters a support-link γ that occurs in the path to it, the computation assigns the value γ to γ. When the recursion unwinds back up that path, it assigns γ to each element in the path until either (1) it encounters γ, or (2) it

encounters a point at which the path diverges, and one of the divergent paths returns the value γ while a different divergent path returns the value μ for some other support-link μ. In case (1), γ has a defeater that is γ^σ-dependent but not dependent on any support-link earlier in the path, so we can compute its value in $\mathcal{HG}_{\sigma}$ by splitting the hypergraph into $\mathcal{HG}_{\gamma^\sigma}$ and $\mathcal{HG}_{[\gamma]^\sigma}$. Once we have computed the value for γ in $\mathcal{HG}_{\sigma}$, we can continue unwinding the recursion back up the path. Case (2) is like the case discussed in the previous paragraph in which γ is λ^σ-dependent. In that case we are not interested in the value of γ in $\mathcal{HG}_{\sigma}$. We may later want the value of γ in $\mathcal{HG}_{\lambda^\sigma}$ or $\mathcal{HG}_{[\lambda]^\sigma}$, but we won't know that until we try to compute the value of λ in the latter. So when we encounter diverging paths, one returning γ and the other returning μ, to the element at the junction we assign either γ or μ depending upon which one occurred earlier in the path to the junction from λ. For instance, in the case in which γ is γ^σ-dependent but also λ^σ-dependent, when the unwinding recursion comes to γ, the value assigned will be λ rather than γ. As the recursion unwinds, that value will be assigned to every element it encounters until it reaches the last support-link μ in the path having a μ^σ-dependent defeater. If λ has a defeater that is λ^σ-dependent then the last element of the path is λ. Otherwise, it is the first support-link μ that has a μ^σ-dependent defeater and was encountered in the course of trying to compute a value for λ. When the value γ is carried all the way back to a defeater for a support-link γ (γ may be λ itself) where the path at γ is either empty or does not contain any support-links assigned as values to defeaters for γ (signifying that γ was the first support-link with a γ^σ-dependent defeater that was encountered by the computation), that signifies that γ has a γ^σ-dependent defeater but is not dependent on any earlier item in the path. Because it is not dependent on any earlier item in the path, its degree of justification will be of use in computing the degree of justification of λ. But to compute the degree of justification of γ the hypergraph must be split. So we split the hypergraph and compute the degree of justification of γ. Along the way, the recursion has assigned and recorded numerical values to all of the γ^σ-independent items whose values are required for ultimately computing a value for γ (and hence for λ) in $\mathcal{HG}_{\sigma}$.

We can illustrate this computation with hypergraph (21) in figure one. Suppose $j_0(A) = j_0(B) = 1$. To compute the degree of justification of the support-link λ, we need the degree of justification of the defeat-link δ_1. The path to this defeat-link is $\langle\lambda\rangle$. To compute the degree of justification of the defeat-link δ_1 we need the degree of justification of its root ~Q, and for that we need the degree of justification of its single support-link γ. Thus far the path is still $\langle\lambda\rangle$. To compute the degree of justification of the support-link γ we need the degree of justification of its root R, and for that we need the degree of justification of the support-link μ. We compute that straightforwardly, recording $\langle$NIL,1$\rangle$ in both (link-justifications μ) and (node-justifications R). To compute the degree of justification of the support-link γ we also need the degree of justification of its defeat-link δ_2. At this point we add γ to the path, obtaining the path $\langle\lambda,\gamma\rangle$. To compute the degree of justification of the defeat-link δ_2, we need the degree of justification of its root Q, and for that we need the degree of justification of its single support-link λ. The path to λ is $\langle\lambda,\gamma\rangle$, so λ is λ-dependent. At this point the recursive computation stops and returns the value λ. Then it unwinds, assigning the value λ for Q, δ_2, γ, ~Q, δ_1, and finally λ. The latter signifies that λ is λ-dependent. The hypergraph is then split, values are computed for P and δ_1 relative to λ (i.e., in $\mathcal{HG}_{\lambda}$), and then the degree of justification 0 is computed for λ. We record this by inserting $\langle$NIL,0$\rangle$ into (link-justifications λ). The path for this computation can be diagrammed as in figure one, where paths are written below the arrows and written above the arrows are the values returned as the recursion unwinds.

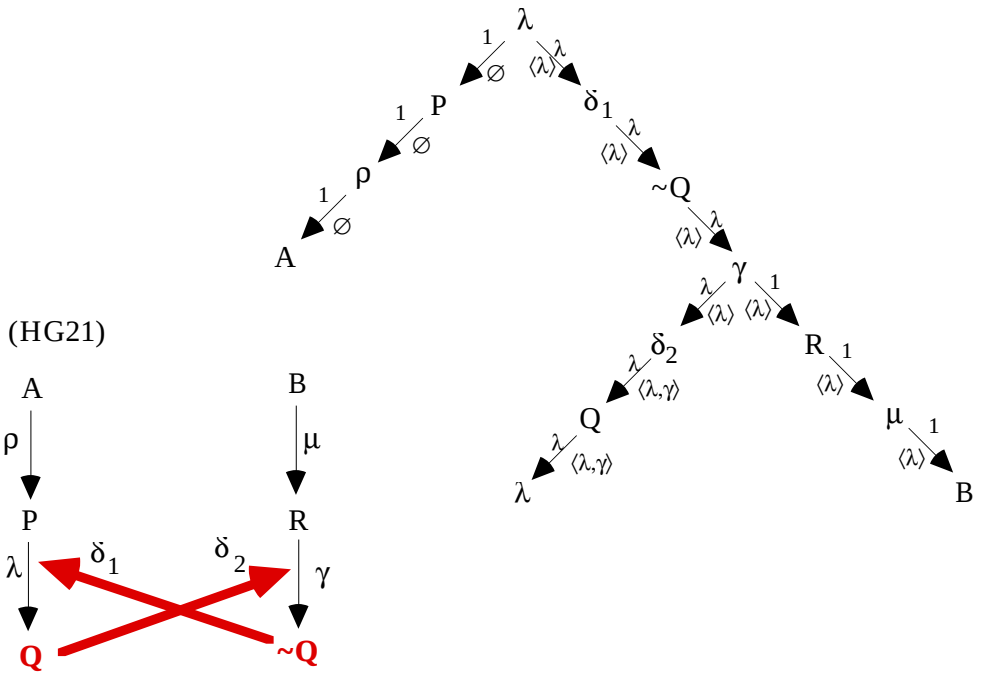


Figure 1. Path of computation in *HG*21.

To illustrate the case of converging paths having support-links as values, consider hypergraph (7). This graph, and the course of the initial computation, is diagrammed in figure 2. In the right fork of the computation, μ receives the value λ rather than ρ because λ precedes ρ in the path ⟨λ,ρ⟩. Similarly, in the left fork, ρ receives the value λ rather than μ because λ precedes μ in the path ⟨λ,μ⟩. The result is that λ receives the value λ, and so the hypergraph is split to produce $HG7_{\lambda}$. In that graph we want to compute the values of λ and ~R. This is diagrammed in figure 3. The computation for λ is trivial. The computation for ~R again leads to cycles for μ and ρ. To compute these degrees of justification, the hypergraph must be slit again. Computing the degree of justification for μ in $HG7_{\lambda}$ leads to the hypergraph $HG7_{\langle\mu,\lambda\rangle}$. In this hypergraph we want to compute the degrees of justification for μ and T_2. These computations are trivial, as diagrammed in figure 4. This allows us to compute a value of 0 for both μ and ρ in $HG7_{\lambda}$. Supplying those values and continuing unwinding the recursion in $HG7_{\lambda}$ produces the course of computation diagrammed in figure 5. That, finally allows us to compute the value 1 for λ in *HG*7.

(HG7)

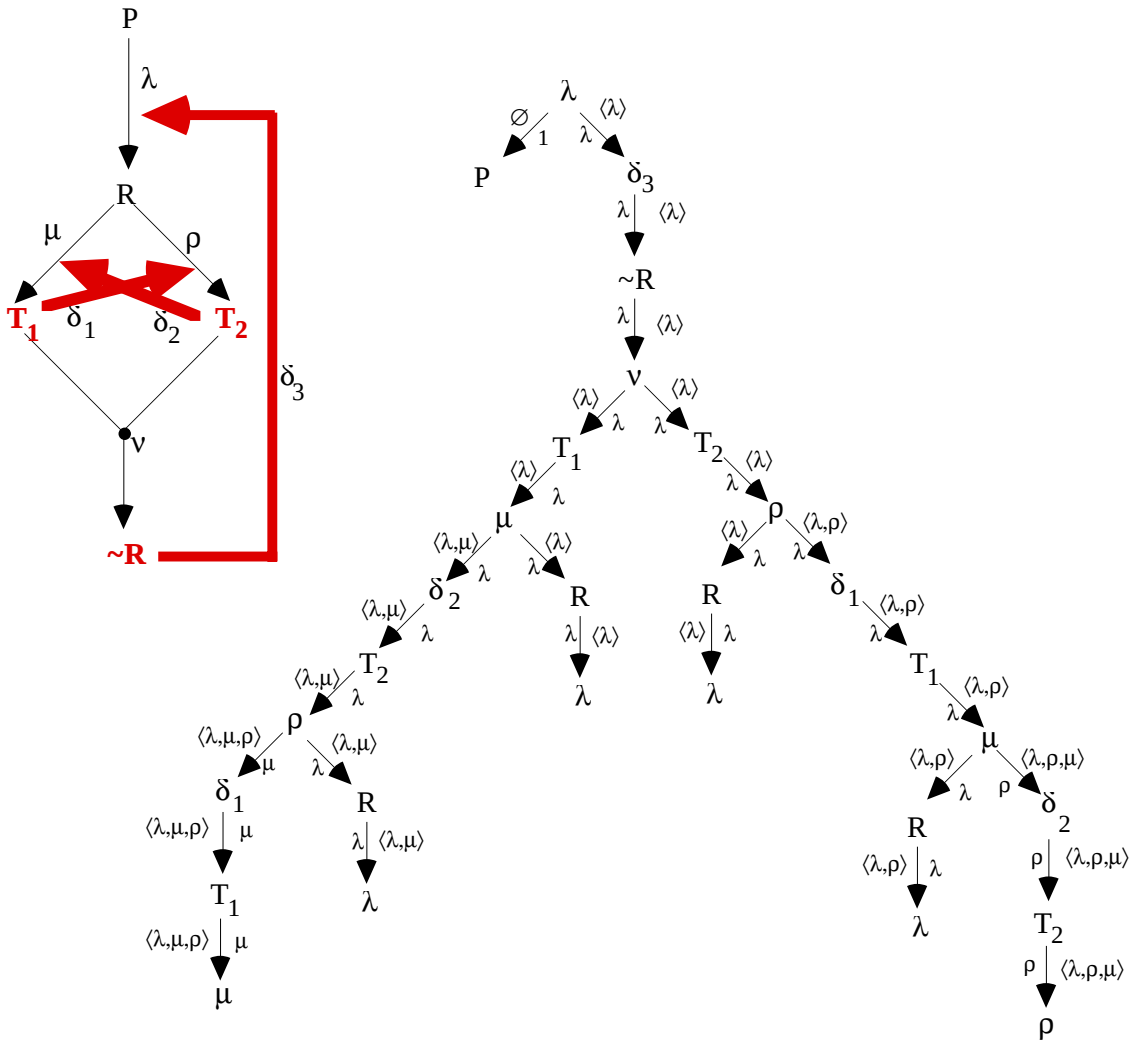


Figure 2. Converging computation paths.

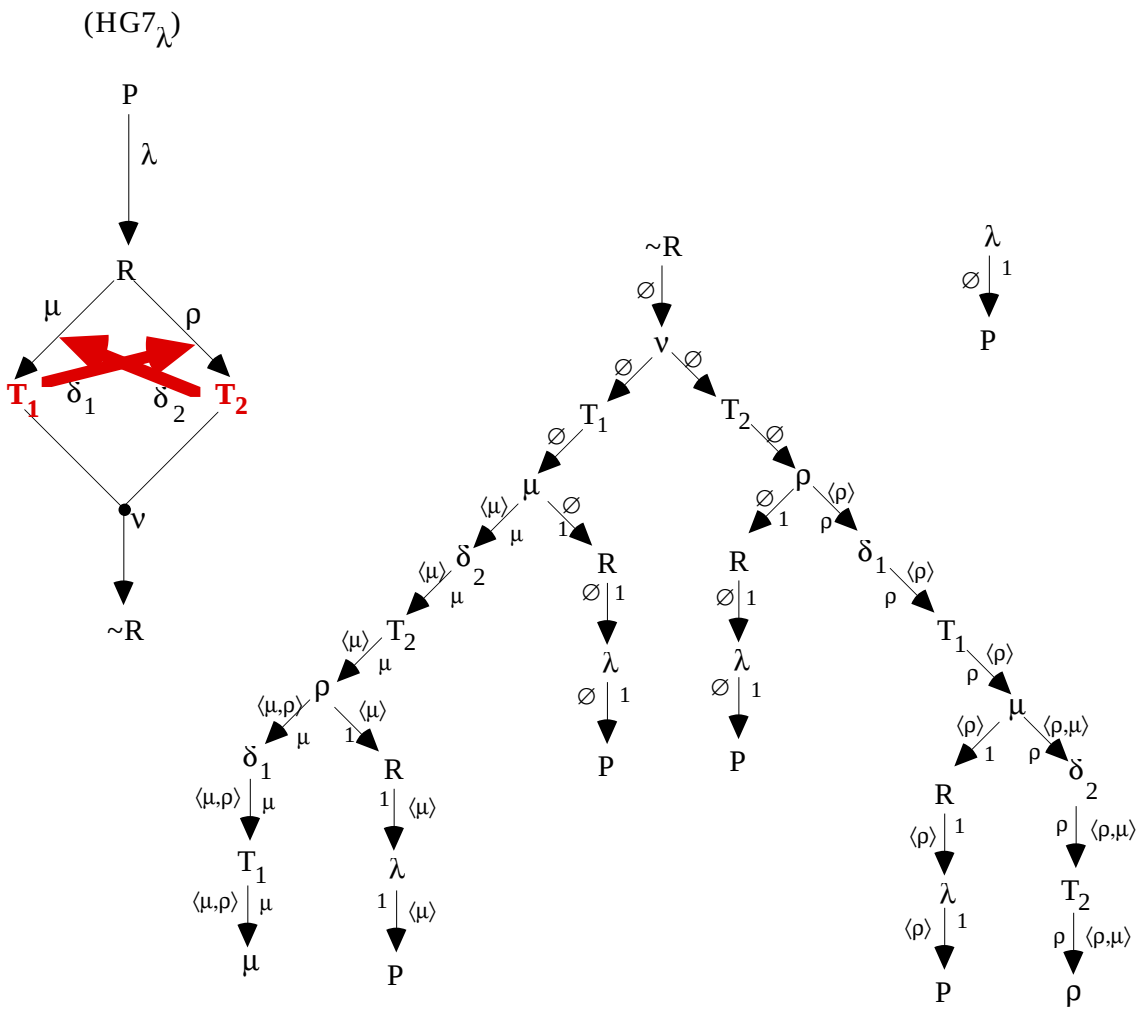


Figure 3: Paths of computation in $HG7_{\lambda}$

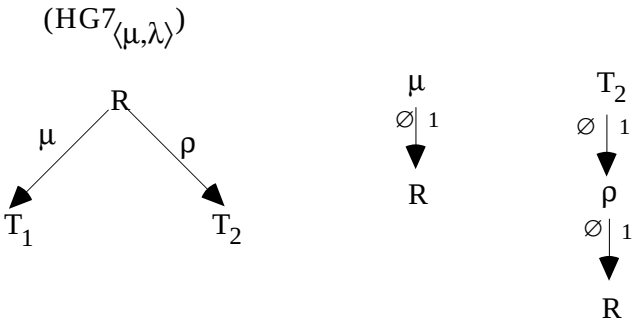


Figure 4. Paths of computation in $HG7_{\langle\mu,\lambda\rangle}$

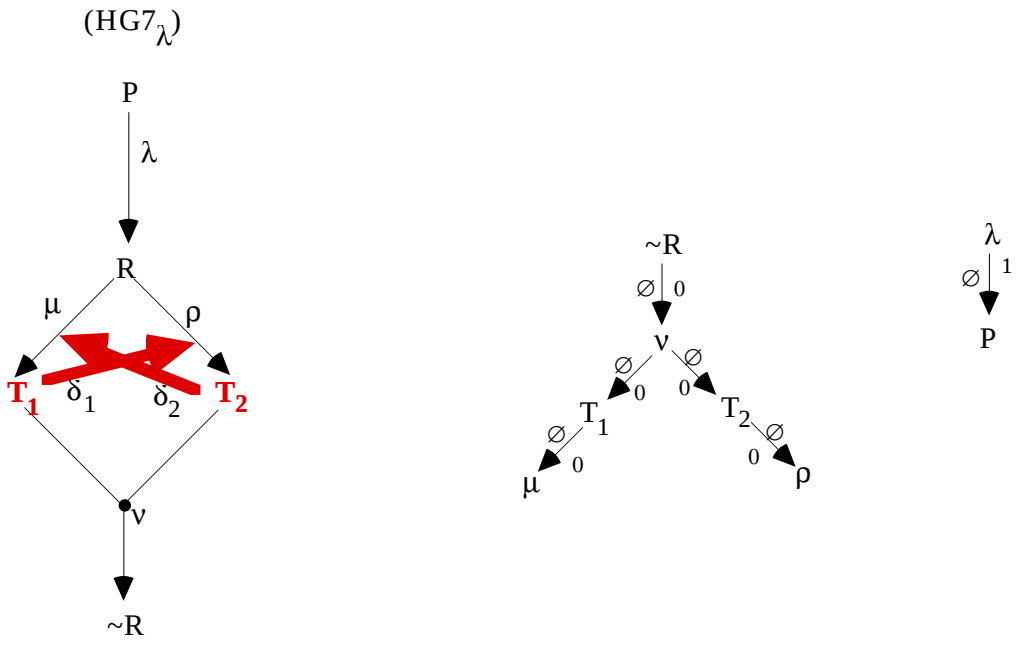


Figure 5. Second path of computation in $HG7_{\lambda}$

Now let us return to *HG*21 and consider another issue. We saw how to compute the degree of justification for λ. Next we want to compute the degree of justification for γ. We will compute the right value if we simply mirror the computation we just performed for λ. However, that will involve recomputing the degrees of justification of some items for which we have already computed values. These include A, B, ρ, μ, P, and R. We would instead like to reuse previously computed values whenever possible. However, when computing the degree of justification for a support-link λ, knowing the previously computed degree of justification of one of its defeat-links δ does not automatically make it useable in the computation. In computing the degree of justification for λ in *HG*, defeat-links are treated differently depending upon whether they are λ-independent or λ-dependent. If δ is λ-independent, then its previously computed degree of justification in *HG* can be used in computing the degree of justification for λ, but if δ is λ-dependent, then what is needed is its value in HG_{λ}. So before reusing a previously computed value for a defeat-link, we need a test for whether it is λ-dependent. This extends to other items as well. If we do not use the previously computed value for δ because it is λ-dependent, then we do not want to use the previously computed value for the root of δ either, because that would just result in the same value being computed again for δ. In general, previously computed values for λ-dependent items cannot be reused in computing the degree of justification for λ.

This can be illustrated by looking again at *HG*21. Suppose we begin by computing, as above, that the degrees of justification for the support-link λ and the inference-node Q are both 0. Next we want to compute the degree of justification for the support-link γ. If we reused the value for either Q or λ we would be led to conclude that the degree of justification for the defeat-link δ_2 is 0, and hence the degrees of justification for γ and ~Q are both 1. But that is incorrect. Their degrees of justification should be 0. The problem is that λ, Q, and δ_2 and all γ-dependent, and hence their values cannot be reused in computing the degrees of justification for γ.

To implement a solution to this problem, the recursive computation must test an item for

λ-dependence. When we have no previously computed values to reuse, λ-dependence is detected by seeing whether the computation returns the value λ. But reusing a value will prevent the recursion from progessing to the point where a circular path is detected. If we are to simplify the computation by reusing values, we need a different way of detecting λ-dependence for previously computed values.

Suppose we have a previously computed degree of justification for a support-link γ, but not λ. We are now computing a degree of justification for λ, we encounter γ, and we want to know whether γ is λ-independent. We can distinguish three ways γ can be λ-dependent. The simplest is that γ might have been inferred from λ. This is captured by the following definition:

Definition: γ is a *link-descendant* of μ iff either the support-link-target of μ is an element of the support-link-basis of γ or the support-link-target of μ is a node-ancestor of an element of the support-link-basis of γ.

```
(defunction link-descendant (L link)
  (or (member (support-link-target link) (support-link-basis L))
      (some #'(lambda (b)   ;; (support-link-basis L)
                (member (support-link-target link) (node-ancestors b)))
            (support-link-basis L))))
```

If γ is a link-descendant of λ then γ is λ-dependent.

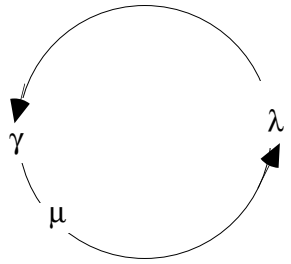


Figure 6. Circular path from γ through λ

Suppose γ is not a link-descendant of λ. We are supposing we encounter γ in the course of computing the degree of justification of λ, so λ is γ-dependent. If γ is λ-dependent, then as λ is also γ-dependent, there is an inference/defeat-path from γ, passing through λ, and returning to γ. This is diagrammed in figure 6. As γ is not a link-descendant of λ, there must be at least one defeat-link on the path between γ and λ. Defeat-links only get on the path by following a support-link for which they are a defeat-link, so there must be a support-link on the path from γ to λ that is followed by a defeat-link for it and that is followed by λ. Let μ be the first such support-link on the path. γ will either be μ itself or a link-descendant of μ. In either case, as μ is the first support-link on the path to be followed by one of its support-links, this path constitutes a circular inference/defeat-path from μ to itself. Thus in following this path when previously computing the value of γ, the computation must have assigned μ to λ. Hence:

If γ has a previously computed degree of justification and γ is not a link-descendant of λ,

then if γ is λ-dependent there is a support-link μ such that either $\gamma = \mu$ or γ is a link-descendant of μ and λ was assigned the value μ in the course of previous computations of degrees of justification.

Conversely, suppose there is a support-link μ such that either $\gamma = \mu$ or γ is a link-descendant of μ and λ was assigned the value μ in the course of previous computations of degrees of justification. If λ was assigned the value μ, there is an inference/defeat-path from μ, passing through λ, and terminating on μ, so μ is λ-dependent. Then as either $\gamma = \mu$ or γ is a link-descendant of μ, it follows that γ is λ-dependent. Therefore:

If γ has a previously computed degree of justification and γ is not a link-descendant of λ, then γ is λ-dependent iff there is a support-link μ such that either $\gamma = \mu$ or γ is a link-descendant of μ and λ was assigned the value μ in the course of previous computations of degrees of justification.

Combining these observations:

If γ has a previously computed degree of justification then γ is λ-dependent iff either (1) γ is a link-descendant of λ, or (2) there is a support-link μ such that either $\gamma = \mu$ or γ is a link-descendant of μ and λ was assigned the value μ in the course of previous computations of degrees of justification.

If we introduce a mechanism for recording the fact that a support-link is assigned another support-link as its value in a computation of degrees of justification, we can use that record to detect λ-dependence. For this purpose we have introduced the field *link-dependencies* in support-links. This consists of a list of pairs $\langle\sigma$,*dependencies*$\rangle$. Whenever a computation of some degree of justification in σ assigns the value γ to a support-link λ relative to some path, we record this by inserting $\lambda\hat{}\sigma$ into the list (link-dependencies γ) and (link-dependencies μ) for every link-descendant μ of γ. This field is set to NIL by RESET-MEMORIES. In computing the degree of justification of λ relative to σ, if we encounter a support-link γ having a previously computed degree of justification relative to σ, and we want to know whether we can reuse that value in computing the degree of justification of λ, we can conclude that γ is $\lambda\hat{}\sigma$-dependent iff either γ is a link-descendant of λ or the $\lambda\hat{}\sigma$ occurs in (link-dependencies γ).

```
;; Where link has a previously computed value, link is sigma-independent and hence that value
;; can be reusing in computing the degree of justification of the first member of sigma. This returns
;; the previously computed value.
(defunction independent-link-value (link sigma)
   (and (not (link-descendant link (car sigma)))
        (not (member sigma (link-dependencies link) :test 'equal))
        (assoc (cdr sigma) (link-justifications link) :test 'equal)))
```

An inference node is $\lambda\hat{}\sigma$-dependent iff it has a $\lambda\hat{}\sigma$-dependent support-link. Thus:

If a node θ has a previously computed value relative to σ, it is $\lambda\hat{}\sigma$-dependent iff either (1) the support-link target of λ is a node-ancestor of θ, or (2) there is a support-link μ such that

support-link-target of μ is either θ or a node-ancestor of θ, and λ was assigned the value μ in the course of previous computations of degrees of justification.

We will record such dependencies in the field *node-dependencies* in inference-nodes. Whenever a computation of some degree of justification in σ assigns the value μ to a support-link λ relative to some path, we record this by inserting λ^σ into the list (node-dependencies N) where N is the support-link-target of μ, and similarly for every node having the support-link-target of μ as a node-ancestor. (This is done recursively by following consequent-links from the support-link-target.) Then in computing the degree of justification of λ relative to σ, if we encounter a node θ having a previously computed degree of justification and we want to know whether we can reuse that value in computing the degree of justification of λ, we can conclude that θ is λ^σ-dependent iff either the support-link target of λ is a node-ancestor of θ, or λ^σ occurs in (node-dependencies γ).

```
;; Where node has a previously computed-value, node is sigma-independent and hence that value
;; can be reused in computing the degree of justification of the first member of sigma. This returns
;; the previously computed-value.
(defunction independent-node-value (node sigma)
   (and (not (member (support-link-target (car sigma)) (node-ancestors node)))
        (not (member sigma (node-dependencies node) :test 'equal))
        (assoc (cdr sigma) (node-justifications node) :test 'equal)))
```

Finally, a defeat-link is λ^σ-dependent iff its root is λ^σ-dependent.

```
;; Where dl has a previously computed-value, dl is sigma-independent and hence that value
;; can be reused in computing the degree of justification of the first member of sigma. This returns
;; the previously computed-value.
(defunction independent-defeat-link-value (dl sigma)
    (independent-node-value (defeat-link-root dl) sigma))
```

The recording of node and link-dependencies is done by the function RECORD-DEPENDENCIES, which is called by COMPUTE-LINK-JUSTIFICATION:

```
;; This records node and link-dependencies discovered during the computation of
;; degrees of justification. It is called by COMPUTE-LINK-JUSTIFICATION. It records the fact
;; that link has been found to be sigma-dependent.
(defunction record-dependencies (link sigma indent)
   (when (and *display?* *j-trace*)
      (indent indent) (princ "RECORDING: ") (princ link) (princ " is ") (princ-sigma sigma)
      (princ "-dependent") (terpri) (indent indent) (princ "RECORDING: ") (princ (support-link-target link))
      (princ " is ") (princ-sigma sigma) (princ "-dependent") (terpri))
   (when (not (member sigma (link-dependencies link) :test 'equal))
      (push sigma (link-dependencies link))
      (when (not (member sigma (node-dependencies (support-link-target link)) :test 'equal))
         (push sigma (node-dependencies (support-link-target link)))
         (dolist (c-link (consequent-links (support-link-target link)))
          (record-dependencies c-link sigma indent)))))
```

This mechanism for testing independence can be illustrated by looking again at *HG*21. Suppose we have already computed the degree of justification of λ, as above. Now we want to compute the degree of justification of γ. In this computation, degrees of justification have already been computed for some elements of the hypergraph, and we want to reuse them rather than recompute

them if they are γ-independent. Figure 7 shows the computation of independence the occurs in the initial computation of values when computing the degree of justification of λ in *HG*21. Figure 8 shows the course of the initial computation in computing the degree of justification of γ. As above, the recursion works its way back to R and then to μ. We have already computed a degree of justification for μ, viz., 1. We want to reuse this value if it is γ-independent. In the course of computing the degree of justification of λ, no circular inference / defeat-path was constructed passing through μ, so no link-dependencies are recorded for μ. Nor is μ a link-descendant of γ, so we can conclude that it is γ-independent and reuse its value. Next the computation seeks a value for the defeater δ_2, and then for λ. We have already computed a degree of justification for λ, viz., 0. We want to reuse this value if it is γ-independent. However, in the course of computing the degree of justification for λ, we traversed a circular inference / defeat-path passing through γ, and so γ is stored as a link-dependency relative to NIL for λ. Accordingly, we can conclude that λ is γ-dependent, and hence we cannot reuse its value. The recursion then moves to δ_2 and then to Q. No value was previously computed for δ_2, but one was computed for Q. However, Q is the support-link-target of λ, so Q is marked γ-dependent, and hence its value cannot be reused. The recursion then continues normally until it finds that γ is γ-dependent, and so the hypergraph must again be split. Once the hypergraph is split, when the computation seeks a value for R relative to γ, it again observes that a previously computed value is recorded in (node-justifications R), viz., ⟨NIL,1⟩. Because this was γ-independent relative to NIL it can be reused in ⟨γ⟩.

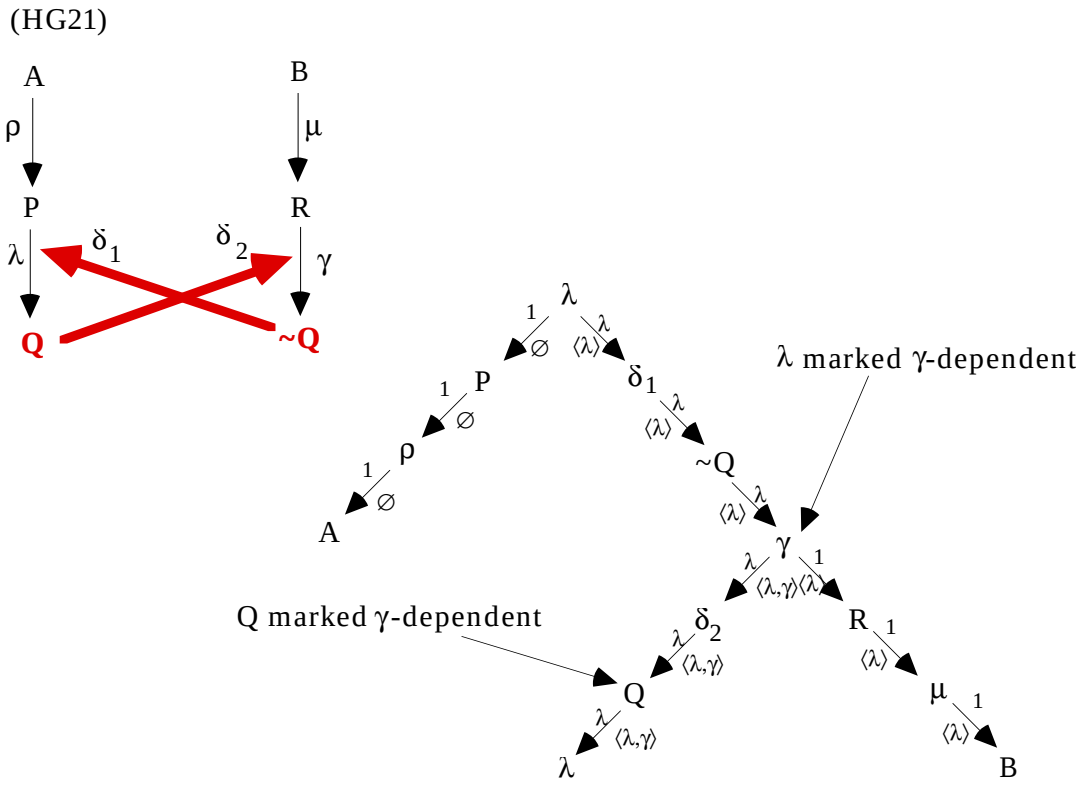


Figure 7. Marking γ-dependence in *HG*21.

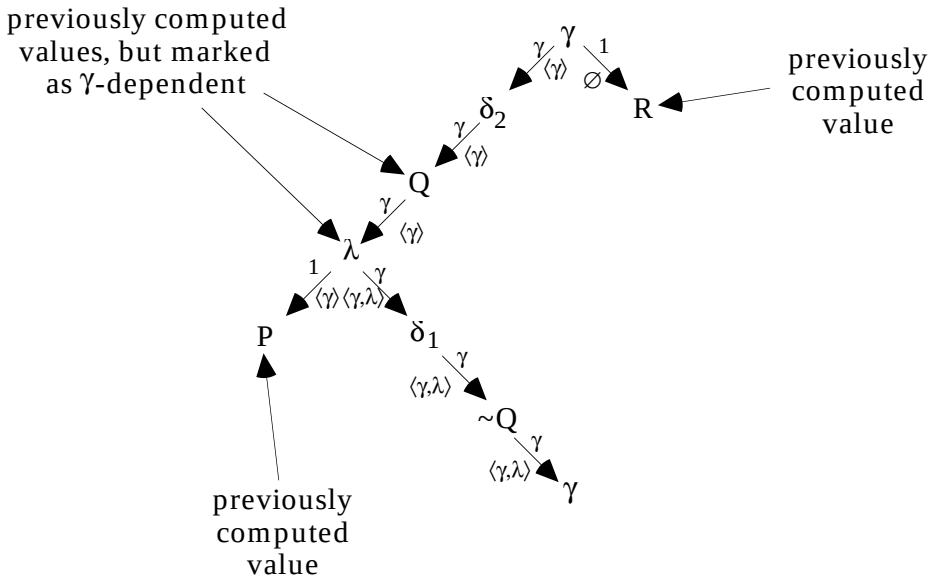
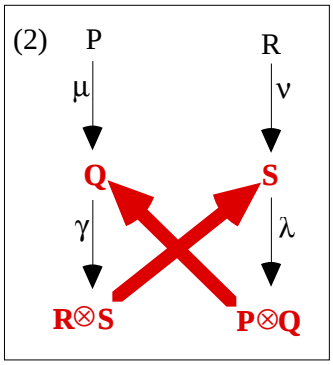


Figure 8. Reusing previously computed γ-independent values.



Suppose we want to compute the degree of justification for a support-link λ. For this purpose, we must compute degrees of justification for the elements of the support-link-basis of λ, and for its defeaters. We can reuse any previously computed values in computing the degree of justification for the basis, but in computing the degrees of justification for the defeaters, we can only reuse previously computed values if they are λ-independent. So the function COMPUTE-LINK-JUSTIFICATION must carry the value of λ through the recursive computation for the defeaters, but not for the basis. To accomplish this, we add a variable *link0* to the functions COMPUTE-NODE-JUSTIFICATION, COMPUTE-LINK-JUSTIFICATION, and COMPUTE-DEFEAT-LINK-JUSTIFICATION. It is set to NIL for computing the degrees of justification for the elements of the support-link-basis of λ, but it is set to λ for computing the degrees of justification for the defeaters for λ. In the latter case, the value of link0 must be carried through the recursive computation to ensure that we do not reuse λ-dependent values. In computing the degree of justification for a basis element, we carry the value NIL through the computation until we come to a support-link γ having a γ-dependent defeater, and then we set link0 = γ and carry that through the computation of the degree of justification for the defeaters for γ. So once link0 is made non-NIL its value is retained throughout

the rest of the computation, but as long as it is NIL, considerations of λ-independent are irrelevant.

To illustrate, consider hypergraph (2). Suppose we want to compute the degree of justification for support-link λ. This support-link has no defeaters, so its degree of justification should be the same as that of its basis S. If that value is already known, we should be able to reuse it, even though S is λ-dependent. If it is not known, we must compute it by computing the degree of justification for ν. ν has a ν-dependent defeater. So in computing the degree of justification of (R ⊗ S) we must carry the identity of ν along to ensure that when we come to a previously computed value we do not reuse it if it is ν-dependent. Thus if we already have a value for Q, we cannot reuse it in computing the degree of justification of ν because it is ν-dependent. But if we already have a value for P, we can reuse that because it is ν-independent. So for computing the degree of justification for the basis of ν, we let link0 remain NIL, but for computing the degree of justification for (R ⊗ S) we set link0 = ν and carry that value throughout the rest of the computation. This is accomplished by the following code.

```
;; path is the reverse of the  list of support-links through whose defeaters the recursion has
;;  traversed before getting to node. Link0 is the support-link initiating the recursive computation
;; leading to node. This returns a support-link or numerical value.
;; Nodes now have a list of values of the form number, (σ . 0). or (σ . path).
(defunction compute-node-justification (node sigma &optional link0 (indent 0) case path)
   (when (and *display?* *j-trace*)
      (cond ((zerop indent) (terpri) (princ "want justification of ") (princ node))
            (t
             (indent indent)
             (cond ((eq case :basis) (princ "need justification of basis element ") (princ node))
                   ((eq case :ind) (princ "need justification of ") (princ node)
                    (princ " as the root of an independent defeat-link"))
                   ((eq case :dep) (princ "need justification of ") (princ node)
                    (princ " as the root of a dependent defeat-link")))
             (when sigma (princ " relative to ") (princ-sigma sigma))
             (when path (princ " and path ") (princ-sigma path))))
      (terpri))
   (let ((value nil))
     (cond
      ;; If the value has already been computed and either link0 = NIL or node is in the support-link-basis
      ;; of link0 or the value is link0-independent, use it
      ((or
        (and
          (or (null link0) (member node (support-link-basis link0)))
          (setf value (cdr (assoc sigma (node-justifications node) :test 'equal))))
        (and link0 (setf value (cdr (independent-node-value node (cons link0 sigma))))))
       (when (and *display?* *j-trace*) (indent indent) (princ "This value has already been computed.") (terpri))
       t)
      ;; Initial nodes get their assigned degrees of justification
      ((eq (node-justification node) :given)
       (when (and *display?* *j-trace*) (indent indent) (princ "This is an initial node.") (terpri))
       (setf value (degree-of-justification node))
       (push (cons sigma value) (node-justifications node)))
      ;;  If the value has been computed in (cdr sigma) and either link0 = NIL or it is link0-independent, use it
      ((and sigma
            (or
             (and (or (null link0) (member node (support-link-basis link0)))
                  (setf value (assoc (cdr sigma) (node-justifications node) :test 'equal)))
             (and link0 (setf value (independent-node-value node (cons link0 (cdr sigma)))))))
       (when (and *display?* *j-trace*) (indent indent) (princ "This node is ") (princ sigma)
             (princ "-independent. It inherits its value from ") (princ-sigma (cdr sigma)) (terpri))
```

```
        (setf value (cdr value))
        (push (cons sigma value) (node-justifications node)))
       (t
        (let ((values nil))
          (dolist (link  (subset #'(lambda (link) (mem sigma (link-in link))) (support-links node)))
           (push (compute-link-justification link sigma link0 (1+ indent) path) values))
          ;;  it receives the maximum value of its sigma-support-links if they all have numerical values.
          (cond ((every #'numberp values)
                 (setf value (maximum0 values))
                 (push (cons sigma value) (node-justifications node))
                 ;; The following is standard stuff from OSCAR_3.33
                 (when (null sigma)
                   (let ((old-value (degree-of-justification node)))
                     (setf (degree-of-justification node) value)
                     (setf (discounted-node-strength node)
                           (if (support-links node)
                             (* (support-link-discount-factor (car (support-links node))) value)
                             value))
                     (when (null old-value) (queue-for-inference node))
                     (setf (old-degree-of-justification node) old-value)
                     (cond
                      ((null old-value)
                        (cond ((not (zerop value)) (push node *new-beliefs*))
                              ((not (member nil (nearest-defeasible-ancestors node)))
                                (push node *new-retractions*))))
                      ((> value old-value) (push node *new-beliefs*))
                      ((< value old-value) (push node *new-retractions*)))
                     )))
                ;;  If any of them are assigned support-links (signifying circularity in the computation)
                ;; then it receives as its value the latest of those support-links in the path.
                (t (setf value (last-link values path)))))))
    (when (and *display?* *j-trace*)
      (indent indent)
      (princ "(justification ") (princ node) (princ " ") (princ-sigma sigma) (princ ") = ") (princ value)
      (when path (princ ", path = ") (princ-sigma path)) (terpri))
    value))

;; This finds the support-link in values that occurs after all other support-links in values in path
(defunction last-link (values path)
  (let ((last-link NIL)
        (remainder path))
    (dolist (value values)
      (when (support-link-p value)
        (let ((new-remainder (member value remainder)))
          (when new-remainder
            (setf remainder new-remainder)
            (setf last-link value)))))
    last-link))

(defunction ~ (x y)
  (if (>= y x) 0.0 (- x y)))

;; path is the reverse of the  list of support-links the recursion has traversed before getting to link.
;; Link0 is the support-link initiating the recursive computation leading to node.
;; This returns a support-link or numerical value.
(defunction compute-link-justification (link sigma link0 &optional (indent 0) path)
  ;  (setf l link l0 link0 s sigma  i indent p path)
  ; (when (eq link (support-link 3)) (setf l link l0 link0 s sigma  i indent p path) (break))
  ;; (step (compute-link-justification l s l0 i p))
```

```
(when (> indent 100) (break))
(when (and *display?* *j-trace*)
   (cond ((zerop indent) (terpri) (princ "want justification of ") (princ link))
         (t
          (indent indent) (princ "need justification of  ") (princ link)
          (when sigma (princ " relative to ") (princ-sigma sigma))
          (when path (princ " and path ") (princ-sigma path))))
   (terpri))
(let ((value nil))
  (cond
   ;; If the value has already been computed and either link0 = NIL or the value is link0-independent, use it
   ((or
     (and (null link0) (setf value (cdr (assoc sigma (link-justifications link) :test 'equal))))
     (and link0 (setf value  (cdr (independent-link-value link (cons link0 sigma))))))
    (when (and *display?* *j-trace*) (indent indent) (princ "This value has already been computed.") (terpri))
    t)
   ;; if path contains link, return link
   ((member link path)
    (when (and *display?* *j-trace*)
     (indent indent) (princ "This path of computation is circular.") (terpri))
    (setf value link))
   ;;  If the value has been computed in (cdr sigma) and either link0 = NIL or it is link0-independent, use it
   ((and sigma
         (or
          (and (null link0) (setf value (assoc (cdr sigma) (link-justifications link) :test 'equal)))
          (and link0 (setf value (independent-link-value link (cons link0 (cdr sigma)))))))
    (when (and *display?* *j-trace*)
     (indent indent)  (princ "This support-link is ") (princ sigma)
     (princ "-independent. It inherits its value from ") (princ-sigma (cdr sigma)) (terpri))
    (setf value (cdr value))
    (push (cons sigma value) (link-justifications link)))
   ;; if (car sigma) is a bracketed-support-link and link is sigma-unsecured, compute the link-justification
   ;; in the critical hypergraph.
   ((and sigma (not (support-link-p (car sigma))) (mem sigma (unsecured-link? link)))
    (let ((sigma-  (cons (caar sigma) (cdr sigma))))
      (when (and *display?* *j-trace*)
       (indent indent)  (princ "This support-link was imported from ") (princ-sigma sigma-) (terpri))
      (setf value (compute-link-justification link sigma- link0 (1+ indent) path))
      (push (cons sigma value) (link-justifications link))))
   ;; if link is not sigma-unsecured, and all defeaters have numerical-values
   ;; then value is the min of the reason-strength and the degrees of justification of elements of
   ;; the basis, less the degree of justification of the undercutting defeater if there is one. The
   ;; defeaters having numerical values signifies that the link is not self-dependent.
   (t
    (block compute-value
     (let ((basis (support-link-basis link)))
       (cond
        (basis
         (let ((D nil) (B nil))
           (cond ((null link0)
                  ;; if link0 = NIL, compute the degrees of justification for the members of the basis
                  ;; without requiring independence, and compute the degrees of justification of
                  ;; the defeat-links requiring link-independence.
                  (dolist (bs basis)
                    (let ((bs-value (compute-node-justification bs sigma nil (1+ indent) :basis path)))
                      (when (eq bs-value 0.0)
                        (setf value 0.0)
                        (push (cons sigma 0.0) (link-justifications link))
                        (return-from compute-value))
```

```
            (push bs-value B)))
          (dolist (dl (support-link-defeaters link))
            (when (member sigma (defeat-link-in dl) :test 'equal)
              (push (compute-defeat-link-justification dl sigma link (1+ indent)  (cons link path)) D)))
          (when (not (every #'numberp D))
            (setf link0 link))
         (t
          ;; if link0 is not NIL, compute the degrees of justification for the members of the basis
          ;;and  the defeat-links requiring link0-independence.
          (dolist (bs basis)
            (let ((bs-value (compute-node-justification bs sigma link0 (1+ indent) :basis path)))
              (when (eq bs-value 0.0)
                (setf value 0.0)
                (push (cons sigma 0.0) (link-justifications link))
                (return-from compute-value))
              (push bs-value B)))
          (dolist (dl (support-link-defeaters link))
            (when (member sigma (defeat-link-in dl) :test 'equal)
              (push (compute-defeat-link-justification dl sigma link0 (1+ indent)  (cons link path)) D)))))
(cond ((and (every #'numberp B) (every #'numberp D))
          (setf value (~ (min (support-link-reason-strength link) (minimum B)) (maximum0 D)))
          (push (cons sigma value) (link-justifications link)))
         ;; if some defeater is assigned a support-link, and the path is either empty or contains
         ;; no support-links assigned as values to defeaters of link, then then link is self-dependent
         ;; and its value is of use in computing the degree of justification of link0. So we split the
         ;; hypergraph and compute its value accordingly.
         ((and (every #'(lambda (sl) (and (not (member sl B)) (not (member sl D)))) path))
          (when (and *display?* *j-trace*)
            (indent indent) (princ link) (princ " is self-dependent") (terpri))
          (let ((independent-defeaters nil)
                (dependent-defeaters nil)
                (sigma1 (cons link sigma))
                (sigma2 (cons (list link) sigma))
                (D1 nil) (D2 nil))
            (split-hypergraph link sigma (1+ indent))
            (dolist (DL (support-link-defeaters link))
              (when (mem sigma (defeat-link-in DL))
                (let ((root (defeat-link-root DL)))
                  (cond ((mem sigma1 (node-in root))
                                (push root dependent-defeaters)
                                (when (mem sigma2 (node-in root)) (push root independent-defeaters)))
                               ((mem sigma2 (node-in root)) (push root independent-defeaters))))))
            (when (not (every #'numberp B))
              (setf B nil)
              (dolist (bs basis)
                (let ((bs-value (compute-node-justification bs sigma link0 (1+ indent) :basis)))
                  (when (eq bs-value 0.0)
                    (setf value 0.0)
                    (push (cons sigma 0.0) (link-justifications link))
                    (return-from compute-value))
                  (push bs-value B))))
            (dolist (d independent-defeaters)
              (push (compute-node-justification D sigma2 link0 (1+ indent) :ind) D1))
            (dolist (d dependent-defeaters)
              (push (compute-node-justification D sigma1 link0 (1+ indent) :dep) D2))
            (setf
             value
             (~ (min (support-link-reason-strength link) (minimum B))
                (+ (maximum0 D1) (maximum0 D2))))
```

```
                        (push (cons sigma value) (link-justifications link))))
                    ;; otherwise, its value is last-link.
                    (t (setf value (last-link (append B D) path))
                       (record-dependencies value (cons link sigma) indent)))))
            (t (setf value (degree-of-justification (support-link-target link)))
               (push (cons sigma value) (link-justifications link))))))))
    (when (and *display?* *j-trace*)
      (indent indent)
      (princ "(justification ") (princ link) (princ " ") (princ-sigma sigma) (princ ") = ") (princ value)
      (when path (princ ", path = ") (princ-sigma path)) (terpri))
    value))

;; path is the reverse of the  list of support-links the recursion has traversed before getting to link.
;; Link0 is the support-link initiating the recursive computation leading to node.
;; This returns a support-link or numerical value.
(defunction compute-defeat-link-justification (DL sigma link0 &optional (indent 0) path)
  (when (and *display?* *j-trace*)
    (cond ((zerop indent) (terpri) (princ "want justification of ") (princ DL))
          (t
           (indent indent) (princ "need justification of ") (princ DL)
           (when sigma (princ " relative to ") (princ-sigma sigma))
           (when path (princ " and path ") (princ-sigma path))))
    (terpri))
  (let ((value nil))
    (cond
     ;; If the value has already been computed and either link0 = NIL or the value is link0-independent, use it
     ((or
       (and (null link0) (setf value (cdr (assoc sigma (defeat-link-justifications DL) :test 'equal))))
       (and link0 (setf value (cdr (independent-defeat-link-value DL (cons link0 sigma))))))
      (when (and *display?* *j-trace*) (indent indent) (princ "This value has already been computed.") (terpri))
      t)
     ;;  If the value has been computed in (cdr sigma) and either link0 = NIL or it is link0-independent, use it
     ((and sigma
           (or
            (and (null link0) (setf value (assoc (cdr sigma) (defeat-link-justifications DL) :test 'equal)))
            (and link0 (setf value (independent-defeat-link-value DL (cons  link0 (cdr sigma))))))))
      (when (and *display?* *j-trace*)
       (indent indent)  (princ "This defeat-link is ") (princ sigma)
       (princ "-independent. It inherits its value from ") (princ-sigma (cdr sigma)) (terpri))
      (setf value (cdr value))
      (push (cons sigma value) (defeat-link-justifications DL)))
     (t
      (let ((root (defeat-link-root DL)))
        (setf value
              (cond ((critical-link DL sigma)
                     ;; critical-defeat-links inherit their value from the noncritical hypergraph
                     (let ((sigma+ (cons (list (car sigma)) (cdr sigma))))
                       (when (and *display?* *j-trace*)
                         (indent (1+ indent)) (princ "This defeat-link was imported from ")
                         (princ-sigma sigma+) (terpri))
                       (compute-node-justification root sigma+ link0 (1+ indent) :dep path)))
                    ;; noncritical-defeat-links have the value of their root
                    (t (compute-node-justification root sigma link0 (1+ indent) :ind path))))
        (when (numberp value) (push (cons sigma value) (defeat-link-justifications DL))))))
    (when (and *display?* *j-trace*)
      (indent indent)
      (princ "(justification ") (princ DL) (princ " ") (princ-sigma sigma) (princ ") = ") (princ value)
      (when path (princ ", path = ") (princ-sigma path)) (terpri))
    value))
```

To illustrate, here is a trace of these functions computing degrees of justification for hypergraph 2, where the support-links and defeat-links are numbered as indicated here:

```
want justification of #<Node 7: (R @ S)>
. need justification of  #<support-link #6 for node 7> relative to nil and path nil
. . need justification of basis element #<Node 6: Q> relative to nil and path nil
. . . need justification of  #<support-link #5 for node 6> relative to nil and path nil
. . . . need justification of #<defeat-link #1 for support-link 5 for node 6> relative to nil and path (<support-link #5>)
. . . . . need justification of #<Node 5: (P @ Q)> as the root of an independent defeat-link relative to nil and path (<support-link
        #5>)
. . . . . need justification of  #<support-link #4 for node 5> relative to nil and path (<support-link #5>)
. . . . . need justification of basis element #<Node 4: S> relative to nil and path (<support-link #5>)
. . . . . need justification of  #<support-link #3 for node 4> relative to nil and path (<support-link #5>)
. . . . . need justification of #<defeat-link #2 for support-link 3 for node 4> relative to nil and path (<support-link #3><support-link #5>)
. . . . . need justification of #<Node 7: (R @ S)> as the root of an independent defeat-link relative to nil and path (<support-link #3><support-link #5>)
. . . . . need justification of  #<support-link #6 for node 7> relative to nil and path (<support-link #3><support-link #5>)
. . . . . need justification of basis element #<Node 6: Q> relative to nil and path (<support-link #3><support-link #5>)
. . . . . need justification of  #<support-link #5 for node 6> relative to nil and path (<support-link #3><support-link #5>)
. . . . . This path of computation is circular.
. . . . . (justification #<support-link #5 for node 6> nil) = #<support-link #5 for node 6>, path = (<support-link #3><support-link #5>)
. . . . . (justification #<Node 6: Q> nil) = #<support-link #5 for node 6>, path = (<support-link #3><support-link #5>)
. . . . . RECORDING: #<support-link #5 for node 6> is (<support-link #6>)-dependent
. . . . . RECORDING: #<Node 6: Q> is (<support-link #6>)-dependent
. . . . . RECORDING: #<support-link #6 for node 7> is (<support-link #6>)-dependent
. . . . . RECORDING: #<Node 7: (R @ S)> is (<support-link #6>)-dependent
. . . . . (justification #<support-link #6 for node 7> nil) = #<support-link #5 for node 6>, path = (<support-link #3><support-link #5>)
. . . . . (justification #<Node 7: (R @ S)> nil) = #<support-link #5 for node 6>, path = (<support-link #3><support-link #5>)
. . . . . (justification #<defeat-link #2 for support-link 3 for node 4> nil) = #<support-link #5 for node 6>, path = (<support-link #3><support-link #5>)
. . . . . need justification of basis element #<Node 2: R> relative to nil and path (<support-link #5>)
. . . . . This is an initial node.
. . . . . (justification #<Node 2: R> nil) = 1.0, path = (<support-link #5>)
. . . . . RECORDING: #<support-link #5 for node 6> is (<support-link #3>)-dependent
. . . . . RECORDING: #<Node 6: Q> is (<support-link #3>)-dependent
. . . . . RECORDING: #<support-link #6 for node 7> is (<support-link #3>)-dependent
. . . . . RECORDING: #<Node 7: (R @ S)> is (<support-link #3>)-dependent
. . . . . (justification #<support-link #3 for node 4> nil) = #<support-link #5 for node 6>, path = (<support-link #5>)
. . . . . (justification #<Node 4: S> nil) = #<support-link #5 for node 6>, path = (<support-link #5>)
. . . . . RECORDING: #<support-link #5 for node 6> is (<support-link #4>)-dependent
. . . . . RECORDING: #<Node 6: Q> is (<support-link #4>)-dependent
. . . . . RECORDING: #<support-link #6 for node 7> is (<support-link #4>)-dependent
. . . . . RECORDING: #<Node 7: (R @ S)> is (<support-link #4>)-dependent
. . . . . (justification #<support-link #4 for node 5> nil) = #<support-link #5 for node 6>, path = (<support-link #5>)
. . . . . (justification #<Node 5: (P @ Q)> nil) = #<support-link #5 for node 6>, path = (<support-link #5>)
. . . . (justification #<defeat-link #1 for support-link 5 for node 6> nil) = #<support-link #5 for node 6>, path = (<support-link #5>)
. . . . need justification of basis element #<Node 1: P> relative to nil and path nil
. . . . This value has already been computed.
. . . . (justification #<Node 1: P> nil) = 1.0, path = nil
. . . #<support-link #5 for node 6> is self-dependent

. . . . Splitting the hypergraph nil relative to #<support-link #5 for node 6>

. . . . Inserting #<support-link #5 for node 6> in (<support-link #5>)
. . . . . Inserting #<Node 1: P> in (<support-link #5>)
. . . . Inserting #<Node 5: (P @ Q)> in (<support-link #5>)
. . . . . Inserting #<support-link #4 for node 5> in (<support-link #5>)
. . . . . Inserting #<Node 4: S> in (<support-link #5>)
. . . . . Inserting #<support-link #3 for node 4> in (<support-link #5>)
```

. . . . . Inserting #<Node 2: R> in (<support-link #5>)
. . . . . Inserting #<support-link #2 for node 2> in (<support-link #5>)

. . . . When sigma = (<support-link #5>):
. . . . . #<support-link #5 for node 6> is in
. . . . #<Node 1: P> is in
. . . . #<Node 2: R> is in
. . . . . #<support-link #2 for node 2> is in
. . . . #<Node 4: S> is in
. . . . . #<support-link #3 for node 4> is in
. . . . #<Node 5: (P @ Q)> is in
. . . . . #<support-link #4 for node 5> is in

. . . . When sigma = ([<support-link #5>]): This hypergraph is empty.

. . . . need justification of basis element #<Node 1: P> relative to nil and path nil
. . . . This is an initial node.
. . . . (justification #<Node 1: P> nil) = 1.0, path = nil
. . . . need justification of #<Node 5: (P @ Q)> as the root of a dependent defeat-link relative to (<support-link #5>) and path nil
. . . . . need justification of #<support-link #4 for node 5> relative to (<support-link #5>) and path nil
. . . . . need justification of basis element #<Node 4: S> relative to (<support-link #5>) and path nil
. . . . . need justification of #<support-link #3 for node 4> relative to (<support-link #5>) and path nil
. . . . . need justification of basis element #<Node 2: R> relative to (<support-link #5>) and path nil
. . . . . This is an initial node.
. . . . . (justification #<Node 2: R> (<support-link #5>)) = 1.0, path = nil
. . . . . (justification #<support-link #3 for node 4> (<support-link #5>)) = 1.0, path = nil
. . . . . (justification #<Node 4: S> (<support-link #5>)) = 1.0, path = nil
. . . . . (justification #<support-link #4 for node 5> (<support-link #5>)) = 1.0, path = nil
. . . . (justification #<Node 5: (P @ Q)> (<support-link #5>)) = 1.0, path = nil
. . . (justification #<support-link #5 for node 6> nil) = 0.0, path = nil
. . (justification #<Node 6: Q> nil) = 0.0, path = nil
. (justification #<support-link #6 for node 7> nil) = 0.0, path = nil
(justification #<Node 7: (R @ S)> nil) = 0.0, path = nil

want justification of #<Node 4: S>
. need justification of #<support-link #3 for node 4> relative to nil and path nil
. . need justification of #<defeat-link #2 for support-link 3 for node 4> relative to nil and path (<support-link #3>)
. . . need justification of #<Node 7: (R @ S)> as the root of an independent defeat-link relative to nil and path (<support-link #3>)
. . . . need justification of #<support-link #6 for node 7> relative to nil and path (<support-link #3>)
. . . . . need justification of basis element #<Node 6: Q> relative to nil and path (<support-link #3>)
. . . . . need justification of #<support-link #5 for node 6> relative to nil and path (<support-link #3>)
. . . . . need justification of #<defeat-link #1 for support-link 5 for node 6> relative to nil and path (<support-link #5><support-link #3>)
. . . . . need justification of #<Node 5: (P @ Q)> as the root of an independent defeat-link relative to nil and path (<support-link #5><support-link #3>)
. . . . . need justification of #<support-link #4 for node 5> relative to nil and path (<support-link #5><support-link #3>)
. . . . . need justification of basis element #<Node 4: S> relative to nil and path (<support-link #5><support-link #3>)
. . . . . need justification of #<support-link #3 for node 4> relative to nil and path (<support-link #5><support-link #3>)
. . . . . This path of computation is circular.
. . . . . (justification #<support-link #3 for node 4> nil) = #<support-link #3 for node 4>, path = (<support-link #5><support-link #3>)
. . . . . (justification #<Node 4: S> nil) = #<support-link #3 for node 4>, path = (<support-link #5><support-link #3>)
. . . . . RECORDING: #<support-link #3 for node 4> is (<support-link #4>)-dependent
. . . . . RECORDING: #<Node 4: S> is (<support-link #4>)-dependent
. . . . . RECORDING: #<support-link #4 for node 5> is (<support-link #4>)-dependent
. . . . . RECORDING: #<Node 5: (P @ Q)> is (<support-link #4>)-dependent
. . . . . (justification #<support-link #4 for node 5> nil) = #<support-link #3 for node 4>, path = (<support-link #5><support-link #3>)
. . . . . (justification #<Node 5: (P @ Q)> nil) = #<support-link #3 for node 4>, path = (<support-link #5><support-link #3>)
. . . . . (justification #<defeat-link #1 for support-link 5 for node 6> nil) = #<support-link #3 for node 4>, path = (<support-link #5><support-link #3>)
. . . . . need justification of basis element #<Node 1: P> relative to nil and path (<support-link #3>)
. . . . . This is an initial node.

. . . . . (justification #<Node 1: P> nil) = 1.0, path = (<support-link #3>)
. . . . . RECORDING: #<support-link #3 for node 4> is (<support-link #5>)-dependent
. . . . . RECORDING: #<Node 4: S> is (<support-link #5>)-dependent
. . . . . RECORDING: #<support-link #4 for node 5> is (<support-link #5>)-dependent
. . . . . RECORDING: #<Node 5: (P @ Q)> is (<support-link #5>)-dependent
. . . . . (justification #<support-link #5 for node 6> nil) = #<support-link #3 for node 4>, path = (<support-link #3>)
. . . . (justification #<Node 6: Q> nil) = #<support-link #3 for node 4>, path = (<support-link #3>)
. . . . RECORDING: #<support-link #3 for node 4> is (<support-link #6>)-dependent
. . . . RECORDING: #<Node 4: S> is (<support-link #6>)-dependent
. . . . RECORDING: #<support-link #4 for node 5> is (<support-link #6>)-dependent
. . . . RECORDING: #<Node 5: (P @ Q)> is (<support-link #6>)-dependent
. . . . (justification #<support-link #6 for node 7> nil) = #<support-link #3 for node 4>, path = (<support-link #3>)
. . . (justification #<Node 7: (R @ S)> nil) = #<support-link #3 for node 4>, path = (<support-link #3>)
. . (justification #<defeat-link #2 for support-link 3 for node 4> nil) = #<support-link #3 for node 4>, path = (<support-link #3>)
. . need justification of basis element #<Node 2: R> relative to nil and path nil
. . This value has already been computed.
. . (justification #<Node 2: R> nil) = 1.0, path = nil
. #<support-link #3 for node 4> is self-dependent

. . Splitting the hypergraph nil relative to #<support-link #3 for node 4>

. . Inserting #<support-link #3 for node 4> in (<support-link #3>)
. . . Inserting #<Node 2: R> in (<support-link #3>)
. . Inserting #<Node 7: (R @ S)> in (<support-link #3>)
. . . Inserting #<support-link #6 for node 7> in (<support-link #3>)
. . . . Inserting #<Node 6: Q> in (<support-link #3>)
. . . . . Inserting #<support-link #5 for node 6> in (<support-link #3>)
. . . . . Inserting #<Node 1: P> in (<support-link #3>)
. . . . . . Inserting #<support-link #1 for node 1> in (<support-link #3>)

. . When sigma = (<support-link #3>):
. . . . #<support-link #3 for node 4> is in
. . #<Node 1: P> is in
. . . . #<support-link #1 for node 1> is in
. . #<Node 2: R> is in
. . #<Node 6: Q> is in
. . . . #<support-link #5 for node 6> is in
. . #<Node 7: (R @ S)> is in
. . . . #<support-link #6 for node 7> is in

. . When sigma = ([<support-link #3>]): This hypergraph is empty.

. . need justification of basis element #<Node 2: R> relative to nil and path nil
. . This is an initial node.
. . (justification #<Node 2: R> nil) = 1.0, path = nil
. . need justification of #<Node 7: (R @ S)> as the root of a dependent defeat-link relative to (<support-link #3>) and path nil
. . . need justification of #<support-link #6 for node 7> relative to (<support-link #3>) and path nil
. . . . need justification of basis element #<Node 6: Q> relative to (<support-link #3>) and path nil
. . . . . need justification of #<support-link #5 for node 6> relative to (<support-link #3>) and path nil
. . . . . need justification of basis element #<Node 1: P> relative to (<support-link #3>) and path nil
. . . . . This is an initial node.
. . . . . (justification #<Node 1: P> (<support-link #3>)) = 1.0, path = nil
. . . . . (justification #<support-link #5 for node 6> (<support-link #3>)) = 1.0, path = nil
. . . . (justification #<Node 6: Q> (<support-link #3>)) = 1.0, path = nil
. . . (justification #<support-link #6 for node 7> (<support-link #3>)) = 1.0, path = nil
. . (justification #<Node 7: (R @ S)> (<support-link #3>)) = 1.0, path = nil
. (justification #<support-link #3 for node 4> nil) = 0.0, path = nil
(justification #<Node 4: S> nil) = 0.0, path = nil

```
want justification of #<Node 5: (P @ Q)>
. need justification of  #<support-link #4 for node 5> relative to nil and path nil
. . need justification of basis element #<Node 4: S> relative to nil and path nil
. . This value has already been computed.
. . (justification #<Node 4: S> nil) = 0.0, path = nil
. (justification #<support-link #4 for node 5> nil) = 0.0, path = nil
(justification #<Node 5: (P @ Q)> nil) = 0.0, path = nil

#<Node 1: P>  --  justification = 1.0
#<Node 2: R>  --  justification = 1.0
#<Node 4: S>  --  justification = 0.0
#<Node 5: (P @ Q)>  --  justification = 0.0
#<Node 6: Q>  --  justification = 0.0
#<Node 7: (R @ S)>  --  justification = 0.0
```

Splitting the hypergraph generates the σ-nodes, σ-links, σ-defeat-links for different choices of σ. Where *HG* is an inference-hypergraph, we define:

Definition: Where σ_1 is a support-link, a node θ is σ-*initial* iff θ has a value computed for σ and θ is either a member of the support-link-basis of σ_1 or θ is σ-independent.

```
;; This assumes that (car sigma) is a support-link.
(defunction initial-node (node sigma)
   (or (and (member node (support-link-basis (car sigma)))
            (assoc sigma (node-justifications node) :test 'equal))
       (independent-node-value node sigma)))
```

Then we define by simultaneous recursion:

Definition:

σ-*nodes*, σ-*links*, σ-*defeat-links*, are defined as follows:

- An inference-node is a ∅-node in *HG* iff it is in *HG*.
- A support-link is a ∅-link in *HG* iff it is in *HG*.
- A defeat-link is a ∅-defeat-link in *HG* iff it is in *HG*.
- Where σ is a sequence of support-links and bracketed support-links and λ is a support-link, let X and Y be the minimal sets satisfying the following conditions:
  - λ is in X, as are all support-links that are λ^σ-critical for ◄⊗λ► or ◄~λ►, and all of the targets of these support-links.
  - All support-links in λ^σ-safe partial-arguments for ⊗λ and ~λ are in Y, as are all of their targets. Any of these support-links that are σ-unsecured are also in X.
  - If a support-link is in X then all members of its basis are in X.
  - If a node is in X, and it is not λ^σ-initial, all of its σ-support-links are in X.
  - If a support-link γ is in X and δ is a σ*-defeat-link for γ that is not λ^σ-critical, δ and Δ(δ) are in X.
  - If a support-link γ is in X and δ is a σ*-defeat-link for γ that is λ^σ-critical, if there are minimal λ^σ-safe partial-arguments for Δ(δ) in *HG*, δ is in X and all support-links contained in the minimal λ^σ-safe partial-arguments are in Y, as are their targets. Any of these support-links that are σ-unsecured are also in X unless they are λ^σ-

independent.

- An inference-node is a λ^σ-node iff it is in X.
- A support-link is a λ^σ-link iff it is in X.
- A defeat-link is a λ^σ-defeat-link iff either (1) it is in X, or (2) its root is in Y and it is λ^σ-critical.
- An inference-node is a [λ]^σ-node iff it is in Y.
- A support-link is a [λ]^σ-link iff it is in Y.
- A defeat-link is a [λ]^σ-defeat-link iff it is in Y.

```
(defunction split-hypergraph (link &optional sigma (indent 0))
   (when (and *display?* *s-trace*)(terpri)  (indent indent) (princ "Splitting the hypergraph ")
         (princ-sigma sigma) (princ " relative to ") (princ link) (terpri))
   (let ((sigma1 (cons link sigma))        ;; the link-critical graph
         (sigma2 (cons (list link) sigma)))    ;; the link-noncritical graph
     ;; add link to the critical-graph and close under the closure conditions
     (add-link-to-critical-graph link sigma1 sigma2 indent)
     ;; for each support-link that is sigma1-critical for the defeat-link-root of a link-defeater of link, add it and
     ;; its target to the critical-graph, and close under the closure conditions.
     (dolist (dl (support-link-defeaters link))
       (dolist (L (critical-support-links dl sigma1))
         (when (not  (member sigma1 (node-in (support-link-target L)) :test 'equal))
           (when (and *display?* *s-trace*)
             (terpri) (indent indent) (princ "Inserting ") (princ (support-link-target L))
             (princ " in ") (princ-sigma sigma1))
           (push sigma1 (node-in (support-link-target L))))
         (add-link-to-critical-graph L sigma1 sigma2 (1+ indent))))
     (when (and *display?* *j-trace*) (display-split link sigma indent))))

;; This adds link to the critical-graph sigma, and closes it under the closure conditions
(defunction add-link-to-critical-graph (link sigma1 sigma2 &optional (indent 0))
   (when (not (mem sigma1 (link-in link)))
      (when (and *display?* *s-trace*)
        (terpri) (indent indent) (princ "Inserting ") (princ link) (princ " in ") (princ-sigma sigma1))
      (push sigma1 (link-in link))
      ;; if a support-link is in sigma1 then all members of its basis are in sigma1
      (dolist (B (support-link-basis link))
       (add-node-to-critical-graph B sigma1 sigma2 (1+ indent)))
      (dolist (dl (support-link-defeaters link))
        (when (and (mem (cdr sigma1) (defeat-link-in dl)) (not (mem sigma1 (defeat-link-in dl))))
       ;; if a support-link is in sigma1 all of its sigma1-noncritical defeat-links and their roots are in sigma1
         (cond ((not (critical-link dl sigma1))
                 (push sigma1 (defeat-link-in dl))
                 (when (and *display?* *s-trace*)
                    (terpri) (indent (1+ indent)) (princ "Inserting ") (princ dl) (princ " in ") (princ-sigma sigma1))
                 (add-node-to-critical-graph (defeat-link-root dl) sigma1 sigma2 (1+ indent)))
               ;; if a sigma1-critical defeat-link has minimal sigma1-safe arguments, it is in sigma1
               ;; and all support-links and their targets in the minimal sigma1-safe arguments
               ;; are in the noncritical-graph sigma2. If one of these support-links is (cdr sigma1)-
               ;; unsecured and it is not sigma1-independent, it is also in the critical-graph sigma1.
               (t  (multiple-value-bind
                       (links unsecured-links)
                       (minimal-safe-arguments sigma1 dl indent)
                       (when links
                          (when (and *display?* *s-trace*)
                            (terpri) (indent (1+ indent)) (princ "Inserting ") (princ dl)
```

```
            (princ " in ") (princ-sigma sigma1))
        (pushnew sigma1 (defeat-link-in dl)))
    (dolist (L links)
        (when (not (mem sigma2 (link-in (support-link-target L))))
            (push sigma2 (node-in (support-link-target L)))
            (when (and *display?* *s-trace*)
                (terpri) (indent (1+ indent)) (princ "Inserting ") (princ (support-link-target L))
                (princ " in noncritical graph  ") (princ-sigma sigma2)))
        (when (not (mem sigma2 (link-in L)))
            (push sigma2 (link-in L))
            (when (and *display?* *s-trace*)
                (terpri) (indent (1+ indent)) (princ "Inserting ") (princ L)
                (princ " in noncritical graph ") (princ-sigma sigma2))))
    (dolist (L unsecured-links)
        (push sigma2 (unsecured-link? L))
        (when (not (independent-link-value L sigma1))
            (add-link-to-critical-graph L sigma1 sigma2 (1+ indent)))))))))))
```

```
;; This adds node to the critical-graph sigma1, and closes it under the closure conditions
(defunction add-node-to-critical-graph (node sigma1 sigma2 &optional (indent 0))
   (when (and (not (member node (support-link-basis (car sigma1))))
              (not (independent-node-value node (car sigma) (cdr sigma))))
     (when (and *display?* *s-trace*)
      (terpri) (indent indent) (princ "Inserting ") (princ node) (princ " in ") (princ-sigma sigma1))
      (push sigma1 (node-in node))
      ;; if a node is in sigma1 and it is not sigma1-independent, all of its support-links are in sigma1
      ;; It is assumed that a node is sigma1-independent iff it already has a value computed in (cdr sigma1)
      (when (not (initial-node node sigma1)))
         (dolist (L (support-links node))
          (add-link-to-critical-graph L sigma1 sigma2 (1+ indent))))))
```

Definition: A support-link γ is *contained in a minimal σ-safe argument* for Δ(δ) iff γ is a σ-subsidiary-support-link for Δ(δ) relative to the empty path.

```
;; This returns the lists of links and unsecured links in the minimal sigma-safe arguments for the root of DL.
(defunction minimal-safe-arguments (sigma  DL)
   (subsidiary-support-links-for-node (defeat-link-root DL) sigma nil DL))
```

Definition:

The σ-*subsidiary-support-links* and σ-*unsecured* support-links are defined recursively:

- If ρ is the empty path, or ρ is a list of support-links and β is in the basis of the first member of ρ, then if β is σ-initial the set of σ-subsidiary-support-links for β relative to ρ is empty. If β is not σ-initial then the set of σ-subsidiary-support-links for β relative to ρ is the union of all the sets of σ-subsidiary-support-links for γ relative to ρ such that γ is a support-link for β, γ^ρ is not σ-convergent, and γ∉ρ. One of these links is σ-unsecured for β relative to ρ iff it is σ-unsecured for one of the γ's relative to ρ
- If γ∉ρ, ρ is not σ-safe, and either γ is σ-independent or γ^ρ is σ-safe, the set of σ-subsidiary-support-links for γ relative to ρ is {γ}, and γ is σ-unsecured.
- If γ∉ρ and γ^ρ is not σ-safe and not σ-convergent:
  - if for some β in the basis of γ, β is not σ-initial and the set of σ-subsidiary-support-links for β relative to ρ is empty, the set of σ-subsidiary-support-links for γ relative to ρ is empty.
  - otherwise, the set of σ-subsidiary-support-links for γ relative to ρ is the result of

adding γ to the union of all the sets of σ-subsidiary-support-links relative to γ^ρ for members of the basis γ that are not σ-initial, and the σ-unsecured links for γ relative to γ^ρ are those links σ-unsecured relative to ρ for some member of the basis of γ that are not σ-initial.

- If γ∉ρ and γ^ρ is σ-convergent then the set of σ-subsidiary-support-links for γ relative to ρ is empty.

The construction of σ-safe arguments occurs in the course of splitting hypergraphs, which is part of the recursive computation of degrees of justification. Because the hypergraph is not split at σ_1 until an attempt is made to compute degrees of justification for everything upon which σ_1 depends, it has already been determined which of the latter are σ-dependent, and so we can test that by seeing whether σ-dependence has been recorded in link-dependencies and node-dependencies.

```
;; This returns the lists of links and unsecured links in the minimal sigma-safe arguments for the root of DL.
(defunction minimal-safe-arguments (sigma  DL indent)
   (cond ((and *display?* *safe-trace*)
          (terpri) (terpri) (indent indent) (princ "Computing minimal ") (princ-sigma sigma)
          (princ "-safe-arguments for ") (princ DL) (terpri)
          (multiple-value-bind
              (links unsecured-links)
              (subsidiary-support-links-for-node (defeat-link-root DL) sigma nil DL (1+ indent))
              (indent indent) (princ "The set of support-links in minimal ") (princ-sigma sigma)
              (princ "-safe-arguments for ") (princ DL) (princ " = ") (princ-sigma links) (terpri)
              (values links unsecured-links)))
         (t (subsidiary-support-links-for-node (defeat-link-root DL) sigma nil DL (1+ indent)))))

;; set of sigma-subsidiary-support-links for node relative to path, where path is an sigma-support-path
;; for (defeat-link-root DL). This returns the two values links and unsecured-links
(defunction subsidiary-support-links-for-node (node sigma path DL indent)
   (when (and *display?* *safe-trace*)
     (indent indent) (princ "want subsidiary-support-links-for ") (princ node)
       (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
       (princ-sigma path) (terpri))
   (let ((links nil) (unsecured-links nil))
     (when (not (initial-node node sigma))
       (dolist (L (support-links node))
        (multiple-value-bind
            (L-links L-unsecured-links)
            (subsidiary-support-links-for-link L sigma path DL (1+ indent))
            (dolist (L-link L-links)
               (when (and (not (member L-link path))
                          (not (convergent-support-path (cons L-link path) sigma DL)))
                  (push L-link links)
                  (when (member L-link L-unsecured-links) (push L-link unsecured-links))))))
       (when (and *display?* *safe-trace*)
        (indent indent) (princ "the set of subsidiary-support-links-for ") (princ node)
        (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
        (princ-sigma path) (princ " = ") (princ-sigma links) (terpri))
       (values links unsecured-links))))

;; set of sigma-subsidiary-support-links for link relative to path, where path is an sigma-support-path
;; for (defeat-link-root DL). This returns the two values links and unsecured-links. It assumes
;; (not (safe-support-path path sigma DL)), because the recursion would have stopped before
;; getting to such a path.
```

```
(defunction subsidiary-support-links-for-link (link sigma path DL indent)
   (when (and *display?* *safe-trace*)
     (indent indent) (princ "want subsidiary-support-links-for ") (princ link)
      (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
      (princ-sigma path) (terpri))
   (when (not (member link path))
      (let ((new-path (cons link path)))
       (cond ((or
            (cond ((and *display?* *safe-trace*)
                    (let ((ind (independent-link-value link sigma)))
                     (when ind
                        (indent indent) (princ-sigma link) (princ " is ") (princ-sigma sigma)
                        (princ "-independent, so") (terpri)
                        (indent indent) (princ "the set of subsidiary-support-links-for ") (princ link)
                        (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
                        (princ-sigma path) (princ " = ") (princ-sigma (list link)) (terpri))
                     ind))
                   (t (independent-link-value link sigma)))
             (cond ((and *display?* *safe-trace*)
                     (let ((safe (safe-support-path new-path sigma DL)))
                      (when safe
                        (indent indent) (princ-sigma new-path) (princ " is a ") (princ-sigma sigma)
                        (princ "-safe support-path, so") (terpri)
                        (indent indent) (princ "the set of subsidiary-support-links-for ") (princ link)
                        (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
                        (princ-sigma path) (princ " = ") (princ-sigma (list link)) (terpri))
                      safe))
                    (t (safe-support-path new-path sigma DL))))
              (values (list link) (list link)))
             ((not (convergent-support-path new-path sigma DL))
              (let ((links nil) (unsecured-links nil))
                (dolist (B (support-link-basis link))
                   (when (not (initial-node B sigma))
                      (multiple-value-bind
                          (B-links B-unsecured-links)
                          (subsidiary-support-links-for-node B sigma (cons link path) DL (1+ indent))
                          (when (null B-links) (return-from subsidiary-support-links-for-link))
                          (setf links (cons link (append B-links links)))
                          (setf unsecured-links (append B-unsecured-links unsecured-links)))))
                (when (and *display?* *safe-trace*)
                   (indent indent) (princ "the set of subsidiary-support-links-for ") (princ link)
                   (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
                   (princ-sigma path) (princ " = ") (princ-sigma links) (terpri))
                (values links unsecured-links)))
             ((and *display?* *safe-trace*)
              (indent indent) (princ-sigma new-path) (princ " is ") (princ-sigma sigma)
              (princ "-convergent, so") (terpri)
              (indent indent) (princ "the set of subsidiary-support-links-for ") (princ link)
              (princ " in ") (princ-sigma sigma) (princ " for ") (princ DL) (princ " relative to path ")
              (princ-sigma path) (princ " = nil") (terpri)
              nil)
             ))))
```

Definition: A defeat-link δ of HG is σ-*critical* in HG iff (1) σ_1 is a support-link λ (not a bracketed support-link), (2) δ lies on a σ-defeat-loop μ in HG.

```
;; DL is a defeat-link and sigma a list of support-links and unit sets of support-links.
(defunction critical-link (DL sigma)
```

```
(when sigma
   (let ((L (car sigma)))
    (when (support-link-p L)
     (let ((critical (assoc sigma (critical-link? DL) :test 'equal)))
       (cond
        (critical (cdr critical))
        (t
         (setf critical
               (some      ;; (defeat-loops L)
                #'(lambda (d-loop)
                    (and (member DL d-loop) (sigma-defeat-loop d-loop sigma)))
                (defeat-loops L)))
         (push (cons sigma critical) (critical-link? DL))
         critical)))))))
```

Definition: A defeat-link δ of *HG* is *hereditarily-σ-critical* in *HG* iff $\sigma \neq \emptyset$ and (1) δ is σ-*critical* in *HG* or (2) δ is hereditarily-σ*-*critical* in *HG*.

```
;; DL is a defeat-link and sigma a list of support-links and unit sets of support-links.
(defunction hereditarily-critical-link (DL sigma)
   (and sigma (or (critical-link DL sigma) (hereditarily-critical-link DL (cdr sigma)))))
```

Definition: A sequence $\langle \delta_1,...,\delta_n \rangle$ of defeat-links is a λ-*defeat-loop* iff (1) the support-link-target of λ is a node-ancestor of the root of δ_1 but not of the root of any δ_k for $k > 1$, (2) λ is the target of δ_n, and (3) for each $k < n$, the ultimate-target of δ_k is equal to or an ancestor of the root of δ_{k+1}, but not of the root of δ_{k+j} for $j > 1$.

```
;; This returns the list of non-circular non-bypassed defeat-paths between links
(defunction defeat-paths (link1 link2 &optional used-links)
   (let ((d-paths nil)
         (target1 (support-link-target link1)))
     (dolist (link (consequent-links target1))
       ;; defeat-paths from the targets of consequent-links of link1 are defeat-paths
       ;; provided link1 does not have the same target as the links used in constructing
       ;; the defeat-path or introduce a bypass.
       (when (and (not (eq link link2)) (not (member link used-links)))
        (let ((target (support-link-target link)))
          (dolist (d-path (defeat-paths link link2 (cons link used-links)))
             (when
                (not
                 (some
                  #'(lambda (DL)
                      (or (eq target (support-link-target (defeat-link-target DL)))  ;; circularity check
                          (member target1 (node-ancestors (defeat-link-root DL)))))  ;; bypass check
                  (cdr d-path)))
                (pushnew d-path d-paths :test 'equal))))))
     (dolist (d-link (supported-defeat-links target1))
       (let* ((link (defeat-link-target d-link))
              (target (support-link-target link)))
         (cond ((eq link link2) (push (list d-link) d-paths))   ;; one-step defeat-path
               ((not (member link used-links))
                ;; if link1 supports the defeat-link d-link, adding d-link to the front of a defeat-path
                ;; from its target to link2 is a defeat-path provided that does not introduce a bypass
                ;; or a circularity.
                (dolist (d-path (defeat-paths link link2 (cons link used-links)))
                   (when
                      (not
```

```
                    (some
                     #'(lambda (DL)
                         (or (eq target (support-link-target (defeat-link-target DL)))  ;; circularity check
                             (member target1 (node-ancestors (defeat-link-root DL)))))  ;; bypass check
                     (cdr d-path)))
                    (pushnew (cons d-link d-path) d-paths :test 'equal)))))))
    d-paths))

;; defeat-loops are remembered in link-defeat-loops
(defunction defeat-loops (link)
  (cond ((eq (link-defeat-loops link) T)
         (setf (link-defeat-loops link) (defeat-paths link link NIL)))
        (t (link-defeat-loops link))))
```

Definition: μ is a σ–*defeat-loop* in *HG* iff μ is σ_1–defeat-loop in *HG* consisting entirely of σ*-defeat-links in *HG*.

```
(defunction sigma-defeat-loop (d-loop sigma)
  (or (null (cdr sigma)) (every #'(lambda (L) (mem (cdr sigma) (defeat-link-in L))) d-loop)))
```

Definition: μ is a σ-*support-path* in *HG* iff μ is a support-path in *HG* and μ consists entirely of σ-support-links in *HG*.

Definition: A σ-support-path ρ for Δ(δ) is σ-*convergent* iff for some support-link λ, $\sigma_1 = \lambda$, and there is a σ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ such that for some i, $\delta = \delta_i$, and (1) if $i = 1$ then $\lambda \in \rho$, and (2) if $i > 1$ then $\text{target}(\delta_{i-1}) \in \rho$.

```
;; path is an sigma-convergent support-path for (root DL)
(defunction convergent-support-path (path sigma DL)
  (let ((L (car sigma)))
    (and (support-link-p L)
         (some    ;; (defeat-loops L)
          #'(lambda (d-loop)
              (and (sigma-defeat-loop d-loop sigma)
                   (let ((d-link (car d-loop))
                         (d-links d-loop)
                         (last-d-link NIL))
                     ;; if DL is the first member of d-loop, L is in path. If DL is a later member of
                     ;; d-loop, and last-d-link is the previous member, (defeat-link-target last-d-link) is in path.
                     (loop
                       (when (eq DL d-link)
                         (cond ((null last-d-link) (return (member L path)))
                               (t (return (member (defeat-link-target last-d-link) path)))))
                       (setf d-links (cdr d-links))
                       (when (null d-links) (return))
                       (setf last-d-link d-link)
                       (setf d-link (car d-links))))))
          (defeat-loops L)))))
```

Definition: A σ-support-path ρ for Δ(δ) is σ-*safe* iff for some support-link λ, $\sigma_1 = \lambda$, and where ρ_1 is the first member of ρ, for each σ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$, if for some i, $\delta = \delta_i$, then (1) if $i = 1$ then $\lambda \notin \rho$ and target(λ) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_1, and (2) if $i > 1$ then $\text{target}(\delta_{i-1}) \notin \rho$ and ultimate-target(δ_{i-1}) is not equal to or a node-ancestor of a member of the support-link-basis of ρ_1.

```
;; path is an sigma-safe support-path for (root DL)
(defunction safe-support-path (path sigma DL)
   (let ((L (car sigma)))
     (and (support-link-p L)
       (every   ;; (defeat-loops L)
        #'(lambda (d-loop)
            (and (sigma-defeat-loop d-loop sigma)
                 (let ((d-link (car d-loop))
                       (d-links d-loop)
                       (last-d-link NIL))
                   ;; if DL is the first member of d-loop, L is not in path and (support-link-target L)
                   ;; is not a node-ancestor of the first member of path. If DL is a later member
                   ;; of d-loop, and last-d-link is the previous member, (support-link-target last-d-link)
                   ;; is not in path and (ultimate-target last-d-link)is not a node-ancestor of the
                   ;; first member of path.
                   (loop
                     (when (eq DL d-link)
                       (cond ((null last-d-link)
                              (return
                               (and (not (member L path))
                                    (not
                                     (some
                                      #'(lambda (b)
                                          (or (eq (support-link-target L) b)
                                              (member (support-link-target L) (node-ancestors b))))
                                     (support-link-basis (car path)))))))
                             (t (return
                                 (and (not (member (defeat-link-target last-d-link) path))
                                      (not
                                       (some
                                        #'(lambda (b)
                                            (or (eq (ultimate-target last-d-link) b)
                                                (member (ultimate-target last-d-link) (node-ancestors b))))
                                       (support-link-basis (car path)))))))))
                     (setf d-links (cdr d-links))
                     (when (null d-links) (return t))
                     (setf last-d-link d-link)
                     (setf d-link (car d-links))))))
        (defeat-loops L)))))
```

Definition: A support-link γ is σ-*critical for* a defeat-link δ iff (1) for some support-link λ, $\sigma_1 = \lambda$, (2) the target of γ is either $\Delta(\delta)$ or a node-ancestor of $\Delta(\delta)$, and (3) there is a λ-defeat-loop $\langle\delta_1,...,\delta_k\rangle$ in σ^* such that for some i, $\delta = \delta_i$, and (3a) if $i = 1$ then either $\gamma = \lambda$ or target(λ) is a node-ancestor of some member of the basis of γ, and (3b) if $i > 1$ then either target(δ_{i-1}) = γ or ultimate-target(δ_{i-1}) is a member of the basis or γ or a node-ancestor of some member of the basis of γ.

```
;; link is an sigma-critical-support-link for DL.
(defunction critical-support-links (DL sigma)
   (let ((L (car sigma))
         (critical-links NIL))
     (when (support-link-p L)
       (dolist (d-loop (defeat-loops L))
         (when (sigma-defeat-loop d-loop sigma)
           (let ((d-link (car d-loop))
                 (d-links d-loop)
```

```
            (last-d-link NIL)
            (link NIL)
            (node NIL))
        ;; if DL is the first member of d-loop, let link be L, node be (support-link-target L).
        ;; If DL is a later member of d-loop, let last-d-link be the previous member and
        ;; let link be its target and node its ultimate-target.
         (loop
           (when (eq d-link DL)
              (cond ((null last-d-link) (setf node (support-link-target L)) (setf link L))
                    (t (setf link (defeat-link-target last-d-link)) (setf node (support-link-target link))))
             (return))
           (setf d-links (cdr d-links))
           (when (null d-links) (return))
           (setf last-d-link d-link)
           (setf d-link (car d-links)))
         (when node
            (let* ((links (cdr (support-links (defeat-link-root DL))))
                   (s-link (car (support-links (defeat-link-root DL)))))
              ;; collect the critical support-links
              (when (eq s-link link) (pushnew s-link critical-links))
              (loop
                (dolist (B (support-link-basis s-link))
                   (when (or (eq node B) (member node (node-ancestors B)))
                      (pushnew s-link critical-links)
                      (setf links (append links (support-links B)))))
                (when (null links) (return))
                (setf s-link (car links))
                (setf links (cdr links)))))))
critical-links))
```