Semantic Nets (a Supplementary Grad Topic)

Semantic networks are a graphical representation of knowledge that can be helpful in formulating efficient algorithms for "taxonomic" reasoning, property inheritance, "semantic priming", making analogies, etc.

Semantic nets and type hierarchies

Relational networks (now often called "knowledge graphs")

Semantic networks were among the earliest devices proposed for representing knowledge (e.g., M. Ross Quillian's work up to 1968, mentioned again later). Early versions tried to capture generic relations such as "plants need water", where "plants" and "water" would be represented as labeled nodes (vertices) and "need" would be a labeled directed edge between them. Later work showed how to extend such networks to represent not only generic or quantified propositions but also particular facts, such as that "Jack are a pizza in the kitchen" (where an event argument is intruduced as well). In recent years, generic semantic networks have been revived under the name "knowledge graphs", with less attention paid to how such networks can be interpreted compared to past work.

Type hierarchies specify a network of *relationships*, namely the relationship of types (like CANARY) to their superordinate types (like BIRD), and the relationship of particular individuals (like FIDO) to their types (like DOG). The *binary predicates* we use to specify these relationships are s (subset of) and e (element of). So a network for a type hierarchy is a *relational network* based entirely on binary predicates (semantically, binary relations). The use of a set-membership relation, e, can be seen as a way of "coercing" a monadic predication like DOG(FIDO) into binary format, FIDO –e–> DOG.

Besides taxonomic relationships, we can also readily represent other binary relationships, such as the relationship of block A being ON block B in a blocks world, or the relationship of JOHN EATing PIZZA3 (a particular pizza). We simply use arcs (edges) with labels such as ON and EAT to connect the entities involved in the relationship, as in the following network fragments:

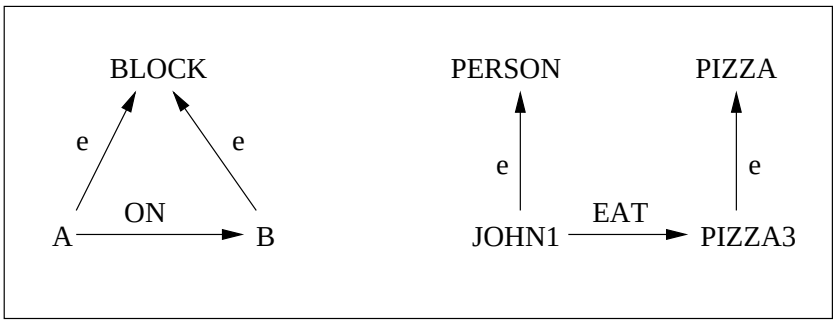


Fig. 2: Binary relations (predicates) as arc labels

This type of relational network has been used in many experimental systems, for instance in Patrick Winston's 1970 program for learning blocks-world concepts such as "house" (a block with a wedge on top) and "arch" (two upright blocks supporting a horizontal block or wedge). An advantage of the network representation for such an application is that it readily suggests *graph-matching* algorithms for finding similarities and differences between the representations of different situations in a simple world such as a blocks world.[1]

But note that in using e (set membership) to relate an individual to a 1-place predicate like DOG, while using an arc label to relate individuals involved in a relation like ON, we are mixing two very different notations. Furthermore, the notation does not extend in any obvious way to 3-place predicates, 4-place predicates, etc. A more uniform alternative, to be explained shortly, is to go from relational to *propositional* semantic networks. But first we will look at an extension of the relational approach based on "breaking down" all predications corresponding to English verbs into a 1-place predication and one or more binary predications.

The idea of this approach is clearest from an example. Consider again "*John ate the pizza*". We can view this sentence as saying that there exists an "eating event", with the added information that the *agent* of the eating is John and that the *theme* (i.e., the thing acted on) is the pizza (say, PIZZA3 as before). This breaks down the original relationship into a type specification and two binary predications as follows:[2]

$\exists$e. EAT(e) $\wedge$ AGENT(e,JOHN1) $\wedge$ THEME(e,PIZZA3)

We can replace the event variable by a Skolem constant, say E5, and then rewrite the

[1] See Winston's "Learning structural descriptions from examples", in P. Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975, 157-209.

[2] The idea of introducing an event variable into the logical translations of verbs is due to Donald Davidson, though he did not suggest breaking down the information in an action sentence as is done here. He would have represented "*John ate the pizza*" as $\exists$e.EAT(e,JOHN1,PIZZA3) (if we ignore the past tense). See D. Davidson, "The logical form of action sentences", in N. Rescher (ed.), *The Logic of Decision and Action*, U. Pittsburgh Press, 1967. Reprinted in D. Davidson & G. Harman (eds.), *The Logic of Grammar*, Dickenson, 1975, 235-245.

above predications by the network relations

E5 –e–> EAT, E5 –agent–> JOHN1, E5 –theme–> PIZZA3.

In linguistics, relations such as AGENT and THEME, connecting an event or situation to the participating entities, are called *thematic roles*, and the representations of sentences based on such roles is referred to as 'neo-Davidsonian'.)[3] One of the advantages of neo-Davidsonian representations of sentences (shared with the original Davidsonian ones) is that they allow us to express many kinds of *modifiers* of such sentences, such as temporal and locative modifiers, in a straightforward way. In particular, we can capture the past tense information in "*John ate the pizza*" by asserting that the eating event, E5, is before the now-point (the time of speech), say NOW34:

E5 –before–> NOW34 .

Further, if we consider the modified sentence "*John ate the pizza in the living room*", we can capture the locative information by asserting that E5 is located in the living room (say, LIVING-ROOM1):

E5 –loc-in–> LIVING-ROOM1 .

Putting all this information in graphical form (and adding type information that JOHN1 is a PERSON, PIZZA3 is a PIZZA, and LIVING-ROOM1 is a LIVING-ROOM), we obtain the following relational semantic net:

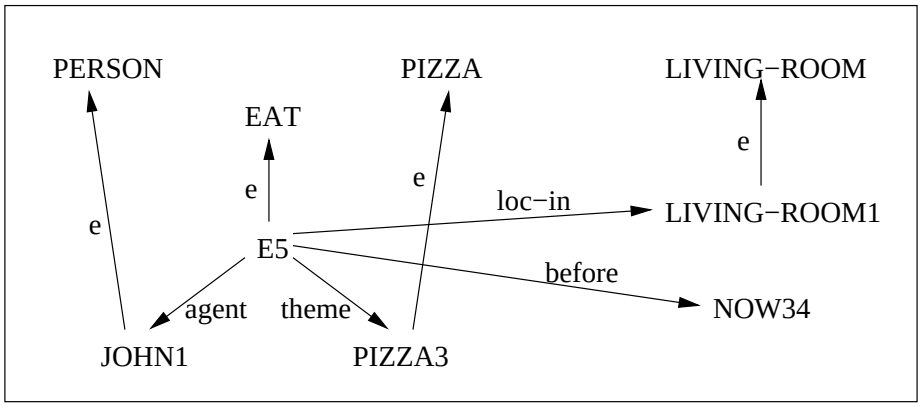


Fig. 3: 'Eating' as a monadic event predicate, with thematic relations (neo−Davidsonian)

The neo-Davidsonian approach easily extends to the representation of verbs with more than two thematic roles, such as "*give*", which normally relates at least 3 enities, as in "*John gave the pizza to Mary*"; here the event type is GIVE, John is still the agent, the pizza is again the theme, and Mary is the *recipient*. (You should have no trouble representing this sentence as a relational semantic net, including the tense information and any modifiers such as "*in the kitchen*" or "*yesterday*". Think of this last modifier as saying that the event is *during* a particular day, and this particular day is *immediately-*

[3]See, e.g., T. Parsons, *Events in the Semantics of English*, MIT Press, 1990.

before an instance of *today*, say, TODAY345.)

However, semantic nets based on neo-Davidsonian sentence representations lose some of their appeal when we try to introduce logical connectives and quantifiers. For example, how should we represent the *negation* of a sentence like "*John ate the pizza*"? This requires saying that there does *not* exist a (past) eating event whose agent is John and whose theme is the pizza. In other words, for every event, either it is not in the past, or it is not of type EAT, or its agent is not John, or its theme is not the pizza in question (PIZZA3).

A simple form of negation that is frequently introduced into relational semantic nets is *negation of arc-labels*. For example, we might use JOHN1 $-\neg e->$ PERSON to express that John is *not* a person, or E1 $-\neg theme->$ PIZZA3 to express that the thing eaten in eating event E1 was *not* the particular pizza denoted by PIZZA3.

Introducing disjunction is more problematic. Note that facts, or propositions, are represented by labeled arcs in these networks. For instance, the fact (or proposition) that John is a person is represented by the arc JOHN1 $-e->$ PERSON. But then to form a disjunction of such facts requires some way of linking together arcs, and saying that these form a disjunction. A notation loosely based on Winston's might be as follows:

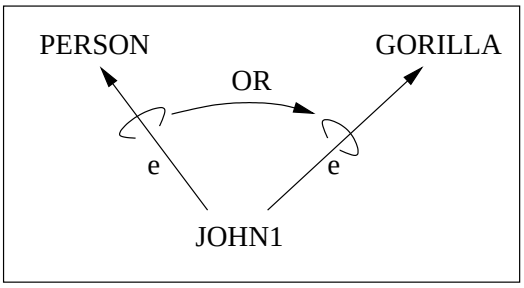


Fig. 4: Disjunction in a relational semantic network

Note that the OR-arc is shown as being directed, but we could make it undirected since disjunction is symmetric in its operands. However, if we also wish to represent other connectives, such as implication, then the direction matters.

At this point we have a "sufficient" set of connectives, in the sense that FOL formulas can always be changed to clause form, and this involves no other connectives except atomic negation and disjunction of literals (in network terms, negated or unnegated arcs). However, we are still lacking quantifiers. Again taking our cue from clause form, we can use *variables* to represent universal quantification. We will use names prefixed with "?" for (universal) variables, to distinguish them from other node names (constants). So for instance to represent

$\forall$x. PERSON(x) $\Rightarrow$ LOVES(x,MARY3),

i.e, "*Everyone loves Mary*" (formalized without introducing events or thematic relations), we could use either of the following relational networks:

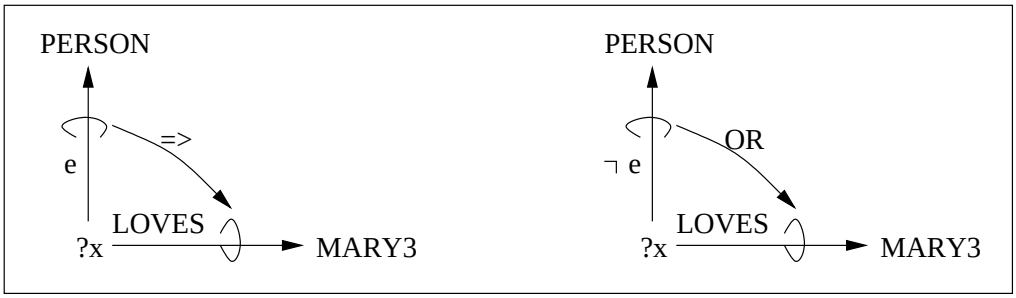


Fig. 5: Representing universal quantification with variables

The network on the left uses implication, and the one on the right disjunction (in effect, clause form) for the same proposition.

Now let's try to represent this in terms of events and thematic relations. Actually, the term "event" is somewhat inappropriate for a static relation like LOVES, so we will broaden our concept of an event to include things that might more properly be called *situations*, *circumstances*, or *eventualities* (e.g., the situation or circumstance of a certain individual loving Mary).[4] First writing this out in FOL, we have

$$\forall x.\ \mathrm{PERSON}(x) \Rightarrow \exists e.\ \mathrm{LOVES}(e) \wedge \mathrm{AGENT}(e,x) \wedge \mathrm{THEME}(e,\mathrm{MARY3}),$$

where we can think of e as the event or situation of x loving Mary. We Skolemize the existential variable,[5] and drop the universal quantifier (rewriting x as ?x):

$$\mathrm{PERSON}(?x) \Rightarrow \mathrm{LOVES}(\mathrm{sk}(?x)) \wedge \mathrm{AGENT}(\mathrm{sk}(?x),?x) \wedge \mathrm{THEME}(\mathrm{sk}(?x),\mathrm{MARY3}).$$

We Skolemized since we don't have a network notation for existential quantifiers ('dependent' on a wider-scope universal quantifier). However, now we have the problem of representing the Skolem term, sk(x), as part of a network. We could represent functional terms as subnetworks by introducing new linkage types, but let us instead allow functional terms directly as node labels. Then the network equivalent of the above proposition is as follows:

[4] *Situation semantics* uses the term "situation" to encompass both event-like and situation-like entities. The term *episode* is used in *episodic logic*, with much the same meaning.

[5] We can Skolemize even though we have not eliminated implications, since the existential quantifier is in a 'positive' environment, i.e., it would remain an existential quantifier even after elimination of implications and distribution of negation.

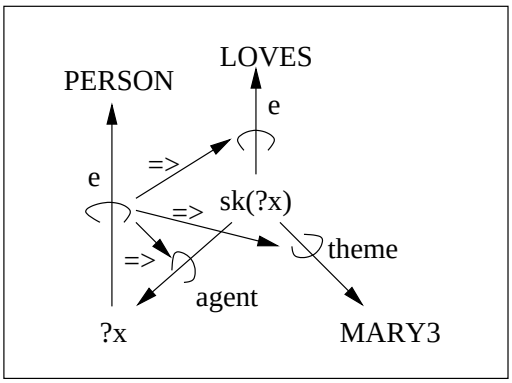


Fig. 6: A Skolem function corresponding to a 'LOVES' situation

Note that we have to use 3 implications (or 3 disjunctions), since we don't have an explicit network notation for conjunction. In effect, the three implications correspond to the 3 clauses we would get in converting the FOL formula to clause form. (You can also think of them as 3 Horn clauses.)

Propositional semantic networks

With the introduction of limited forms of logical connectives and quantification, we are 'pushing the limits' of the relational network notation. We still have no representation for ternary (etc.) relations, except by breaking them down using thematic roles; we have no negation of compound formulas (just atomic formulas), no disjunction of more than two propositions (try it!), no existential quantifiers and no network representation of functional terms.

There are certainly various specialized applications that require no more than the relational network constructs (perhaps without disjunction and universal quantification). Nonetheless, if we want to develop a semantic network representation that is as flexible as FOL, we would like a general way of representing arbitrary FOL formulas as networks. This is what *propositional semantic networks* provide.

Roughly speaking, the 'translation' from FOL to propositional semantic networks is simply this:

Given any FOL expression ϕ (wff, term, or atom),

1. Create a node for ϕ, provided none exists as yet;
2. If ϕ is not an atomic symbol, then
   (a) create nodes for the top-level subexpressions of ϕ, where none exist as yet; this will include a node for the top-level operator of the expression, such as a predicate, function, logical connective, or quantifier;

(b) insert directed edges from the node for ϕ to the nodes for the top-level subexpressions of ϕ; label these edges in some systematic way to make clear what the syntactic roles of the subexpressions are, in relation to ϕ;
(c) if any of the subexpressions of ϕ are again non-atomic, expand them into networks in the same way as described for ϕ.

For example, consider the formula LOVES(JOHN1,MARY3). To represent this as a propositional network, we create a proposition node for the formula (proposition) as a whole. Then we create nodes for LOVES, JOHN1 and MARY1 (if none exist yet), since these are the top-level subexpressions. Finally, we add directed edges from the proposition node to the three subexpression nodes. We label these in some systematic way to indicate what syntactic roles they play in the proposition; for instance, we might use pred, arg1, arg2 respectively to indicate that LOVES is the predicate of the proposition, and JOHN1 and MARY3 are the first and second argument respectively. The result is this small propositional network:

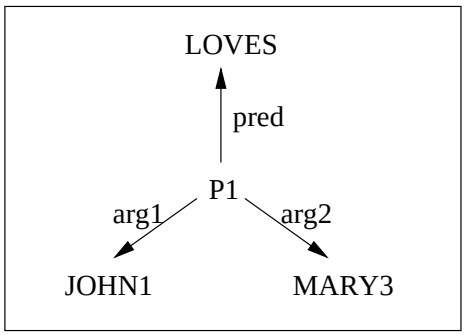


Fig. 7: A simple propositional semantic net

Note that we have arbitrarily labelled the node for the proposition as a whole "P1", just so we have an unambiguous way of referring to that node. This network is rather like some of the relational nets we have seen, but it is crucial to appreciate the differences in semantics. Unlike the e (membership) relation in relational nets, the pred label is merely a *syntactic* indicator, and the same goes for arg1 and arg2. In fact, in a propositional semantic net *none of the edge labels are semantically interpreted – only the nodes receive semantic values.*

The semantic values of the nodes are just what you would expect from the close correspondence between FOL syntax and the network syntax: LOVES is interpreted as a binary relation (a set of ordered pairs), and JOHN1 and MARY3 are interpreted as individuals in the domain of discourse. The proposition node, P1, is either true or false in any given model.

In the same way, we can map more complex formulas into propositional networks, following the above procedure. The two networks below encode "Everyone loves Mary", based on the FOL formula

$\forall$x. PERSON(x) $\Rightarrow$ LOVES(x,MARY3),
and "Everyone loves someone", based on the FOL formula
$\forall$x. PERSON(x) $\Rightarrow \exists$y. PERSON(y) $\wedge \exists$e. LOVES(x,y,e),
after Skolemization to
PERSON(?x) $\Rightarrow$ PERSON(sk1(?x)) $\wedge$ LOVES(?x,sk1(?x),sk2(?x)).

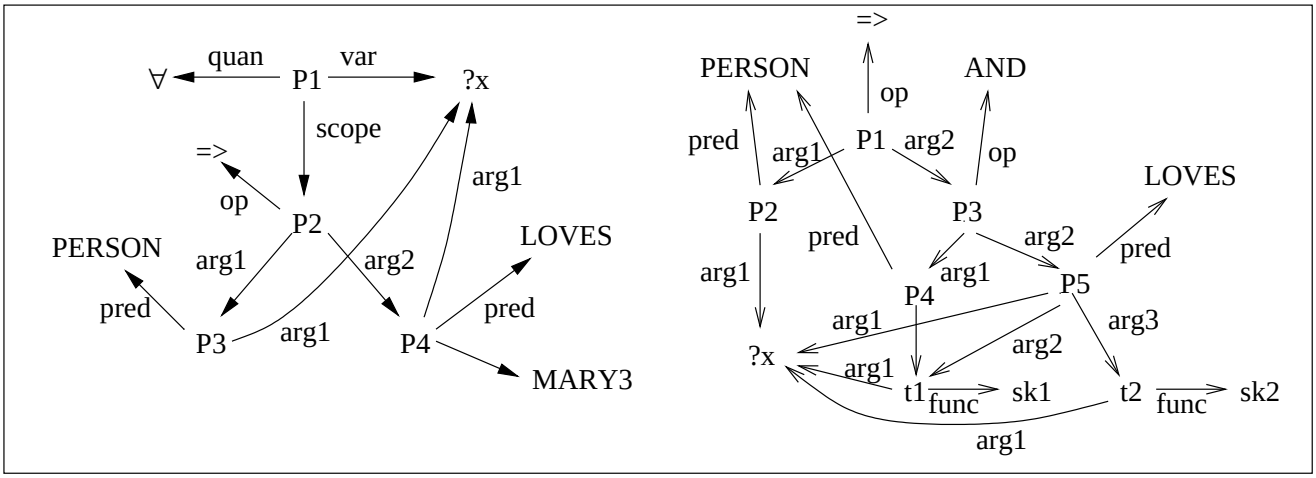


Figure 8: Propositional networks for "Everyone loves Mary" (with explicit $\forall$), and "Everyone loves someone"

Note two differences in the kinds of representations that have been used in the two nets (just to illustrate some of the possibilities). The network on the left ignores situations/ events, while the one on the right includes an explicit Davidsonian event argument; and the network on the left includes an explicit quantifier, while in the network on the right quantification is implicit in the variable node ?x. These differences reflect differences in the original FOL formulas.

These rather complex networks may make you wonder whether there is any point in a network representation. After all, given the way the networks were obtained, they are just another way to represent the expression-subexpression structure of FOL (or any other logical formalism based on expressions)! The answer is that as far as propositional *content* is concerned, FOL and propositional networks are indeed equivalent. So from that perspective, it is a matter of taste which representation we use. The FOL representation is really the more practical, since it is more concise and much easier to write down and read.[6]

However, as pointed out before, graphical representations can be very helpful in the formulation of data structures and algorithms appropriate for some kinds of knowledge

[6]For further discussion of the relation between logic and networks, see L. Schubert, "Extending the expressive power of semantic networks", in *Artificial Intelligence 7*, 163-198; and "Semantic nets are in the eye of the beholder", in J. Sowa (ed.), *Principles of Semantic Networks*, Morgan Kaufmann, 1991, 95-107.

retrieval and inference tasks, One idea that networks suggest is that we should be able to travel not only from propositions to their parts and participants, but also in the "backward" direction, from the entities referenced to the propositions about them. So by allowing bidirectional edge traversal, we can find all the explicitly known facts about any entity by going to the unique node for that entity, and travelling "backward" along the incident links to proposition nodes. This can be useful for 'object-oriented information retrieval', e.g., for responding to a question such as "*What do you know about John*", or "*What does John look like?*".[7]

Another idea, going back at least to the sixties, is to use signal propagation along the edges in a semantic net to determine how closely related two concepts are: the sooner the signals from one arrive at the other, the more closely related they are likely to be. M.R. Quillian used this idea to do some early work on disambiguation. For instance, in a sentence like "John watered the plant", he suggested that we choose the 'vegetable' sense of "plant" in favor of the 'industrial plant' sense, because there are shorter semantic net pathways connecting the former sense to a sense of "watering" which means providing water as nutrient to a plant.[8]

The above uses of semantic nets could be called "associative" processing; i.e., they let us find associations between entities (objects, types, propositions, etc.). In the next subsection we look at techniques that can more properly be called *inference* techniques.

Inference in semantic nets

Transitivity inference

A particularly simple and natural kind of inference is *transitivity inference* of the sort mentioned at the beginning. If there is a directed path from a token node A to a type node P, with successive edge labels $e\,s\,s\,...\,s$, we can infer A–e–> P; and if there is a directed path from type node P to type node Q, where all edge labels are s, then we can infer P–s–> Q.

Normally we would not add such inferences to the network, since this could 'clutter' a network containing n facts with $O(n^2)$ new arcs (each representing a fact inferred by transitivity). However, we can easily make the required inferences whenever the desired result is posed as a query, or arises as a subgoal in some inference task. In the following we focus on query-answering. When we are using a network representation of facts, it is natural to represent the queries as networks as well. For example, the following network corresponds to the query "*Is Fido a mammal?*", which can easily be answered with the network in Fig. 1, using transitivity inference.

[7] For the latter query, it is useful to have a classification of the propositions accessible from each object; for instance, propositions about an object's outward appearance might comprise a separate category (e.g., see J. DeHaan and L.K. Schubert, "Inference in a topically organized semantic net", AAAI-86, 334-338.

[8] M. Ross Quillian, "Semantic memory", in Marvin Minsky (ed.), *Semantic Information Processing*, MIT Press, 1968, 216-270.

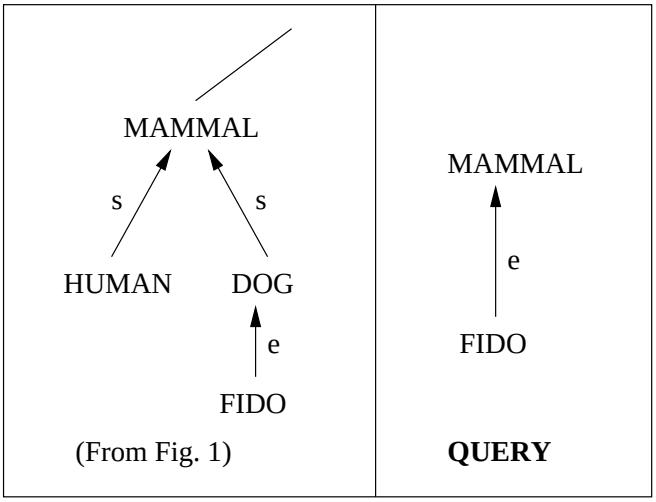


Figure 9: A simple transitive query

Network matching

Another simple type of query-answering can be accomplished by *network matching*. For example, consider questions such as

Did a pizza get eaten?
Where did John eat the pizza?

in relation to Fig. 3. The logical form of these queries, using thematic roles as in Fig. 3, is

? ∃x. PIZZA(x) ∧ ∃e. EAT(e) ∧ THEME(e,x)
? ∃x. ∃e. EAT(e) ∧ AGENT(e,JOHN1) ∧ THEME(e,PIZZA3) ∧ LOC-IN(e,x)

(We put the question mark in front to indicate that this 'operator' operates on the entire formula.) As query networks, these are

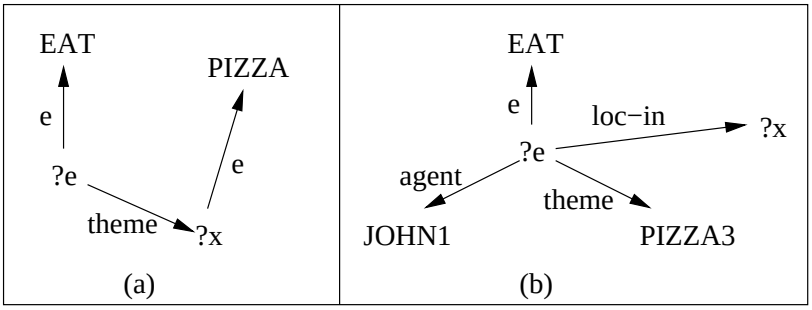


Figure 10: Two queries that can be answered for figure 3

We can easily see how these graphs can be matched against the network in Fig. 3, allowing affirmative answers. The matching process can be viewed as a kind of *unification* process, in which variable nodes become unified with other nodes. Note that the pizza in the first query is represented by variable node ?x, and the eating event by variable node ?e; ?x becomes bound to PIZZA3, and ?e to eating event E5, in Fig. 3. Similarly the

eating location and the eating event in the second query are represented by variable nodes ?x and ?e, and these become bound to LIVING-ROOM1 and E5 respectively in the matching process.

Here the variables in the queries correspond to *existentially* quantified variables in the logical form. This is the opposite of clause form, where variables are regarded as *universally* quantified. We'll comment further on that shortly. Note also that from a resolution perspective, matching a query graph against a given network may correspond to multiple resolution steps. If we had answered the first pizza-query by resolution, we would have negated it and converted to clause form, obtaining

$\neg$PIZZA(x) $\vee \neg$EAT(e) $\vee \neg$THEME(e,x).

Now, the network in Fig. 3 can be viewed as containing the clauses

PIZZA(PIZZA3), EAT(E5), THEME(E5,PIZZA3),

among others, and these together with the denial clause obviously yield the empty clause in three steps. The graph-matching process closely parallels this resolution refutation. However, we are working with the positive form of the query, rather than its denial, and so conjunctions in the query do not get converted to disjunctions.

Network matching plus transitivity inference: Property inheritance

Answering a query may of course require more than one step, as in logical deduction in general. Consider the question

Does John love Mary?

Since we have considered several different network notations, there are several ways to formulate this as a network query. First, if we keep LOVES as a binary predicate (rather than using thematic roles), and use a relational network notation, then the query network is simply as in Fig. 11(a) (cf., Fig. 5). If we decompose the question using thematic relations, then the resultant relational query network is as in Fig. 11(b) (cf., Fig. 6). Finally, if we use a propositional semantic network syntax (and do not decompose into thematic roles), we obtain the network in Fig. 11(c) (cf., Fig. 8).

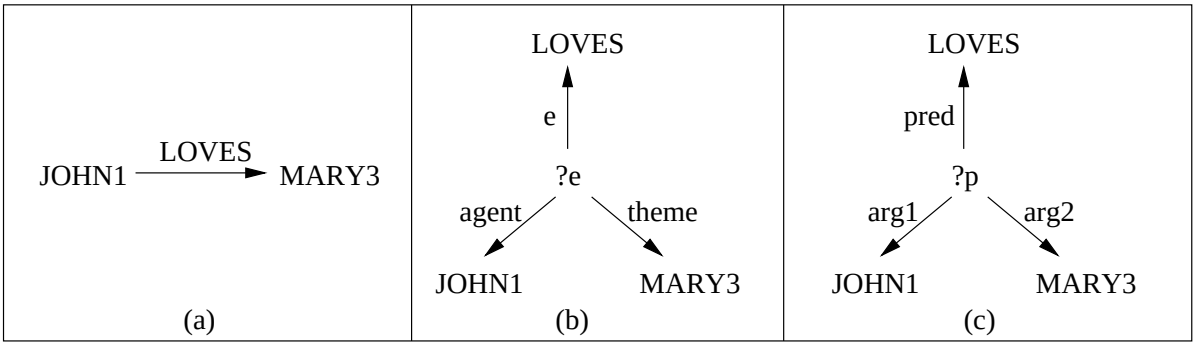


Figure 11: Three ways of asking 'Does John love Mary?" (see figs. 5, 6, and 8 respectively)

We can attempt to answer the question, in any of these forms, by matching the corresponding queries against the networks in Figs. 5, 6, and 8. Let's begin with query format 11(a), matching this against Fig. 5 (left-hand network). We succeed, unifying ?x with JOHN1. However, we are not done because the edge we have matched, ?x–LOVES–> MARY3, *is part of a logical compound*, namely an implication. In other words, ?x–LOVES–> MARY3 is not asserted by the network, but rather an implication is asserted that has ?x–LOVES–> MARY3 as its consequent. We can "reason backward" in this implication, forming the new goal JOHN1—e–> PERSON. (This is the antecedent of the implication, with the unifier applied to it.) Clearly, if we had this type information explicitly in the network, we would succeed in one further matching step. Now it may be that we have the following information as part of a type hierarchy: JOHN1—e–> MAN—s–> PERSON. In that case, we can solve our subgoal with a simple transitivity inference.

Much the same kind of query matching, with creation of a subgoal, would occur in matching query 11(b) against the network in Fig. 6, or query 11(c) against the first network in Fig. 8. In each case, we could complete the answering process if we had a type hierarchy containing the fact that John is a man, and that men are persons.

When answering a query involves use of a type hierarchy as in the above examples, we are in effect making use of *property inheritance*. In general, property inheritance refers to the transfer of properties from a higher node to a lower node in a type hierarchy. In the above example, we used the fact (encoded in 3 variant networks) that every PERSON loves Mary to infer that JOHN1 loves Mary. So we transferred the property of loving Mary from the higher-level PERSON node to the lower-level token node, JOHN1. Because of the ease of checking hierarchy relationships in a semantic net, a semantic net representation is considered particularly appropriate for applications emphasizing property inheritance. (However, it should be pointed out that we can get the same effect with, say, a clausal reasoning system that uses a type specialist to do type checking. In such a case, it might be appropriate to view the type specialist, but not the clausal knowledge base as a whole, as a semantic net.)

Now let's look a little more closely at the representation of queries in quantifier-free form. As already noted, it is the *existentially* quantified variables that are retained as variables in the query. In fact, in general the proper way to eliminate quantifiers in a query (or goal to be proved) involves *reverse Skolemization*: we replace universally quantified variables by Skolem constants and functions, where the arguments of a Skolem function are all the existentially quantified variables in whose scope it lies. That this makes sense is clear from the analogy to resolution refutations: even though we are not using the denial of the query, we want unification to work just as in the case of a resolution refutation, and so we Skolemize *as if* we had negated the query. For example, the question

"*Is there someone that everyone loves?*", i.e.,

? ∃y. PERSON(y) ∧ ∀x. PERSON(x) ⇒ LOVES(x,y),

becomes

? PERSON(y) ∧ (PERSON(F(y)) ⇒ LOVES(F(y),y)).

If we convert this to a network and match it against the network in Fig. 5 (on the left), we succeed for implicative part of the query, unifying ?y with MARY3 and F(MARY3) with ?x. It then remains to verify PERSON(MARY3), represented in network form as MARY3 —e–> PERSON, which could again be done with a suitable type hierarchy.

We should note that propositional semantic nets may not make property inheritance quite as easy as relational nets. The 'algorithm' given for generating propositional semantic networks from FOL representations does not immediately guarantee the conversion of type relations into a simple network format. After all, the FOL encoding of a statement such as that all dogs are mammals would be a quantified implicative formula like ∀x. DOG(x) ⇒ MAMMAL(x). This translates into a rather complex propositional network fragment. We can simplify type relationships by augmenting FOL to allow statements such as SUB(DOG,MAMMAL) (DOG is a subtype of MAMMAL). We would treat SUB as having a fixed semantics (namely, as expressing the subset relation), independent of the chosen interpretation. This would now lead to quite a simple network fragment as well, and property inheritance would then be as easy as in relational semantic nets.

Nonmonotonic inheritance

Finally, we should take our first look at a type of *nonmonotonic reasoning*. We cannot always express our general knowledge about types of objects in the world as universal statements. For example, it seems true to say that canaries fly (or can fly), yet we know there are exceptions, such as fledgling canaries, canaries with clipped wings, and perhaps some genetically abnormal canaries.

We could represent this 'generic' (but not universal) property of canaries by some new type of link, which will be notated here as an *unlabelled* arc. (This is rather appropriate,

since we're not sure what such links mean!)

CANARY ——> FLY

We read this as "Generally, canaries fly", or "By default, canaries fly" if we want to emphasize the fact that this is not a universal statement. Now if we also have

TWEETY —e–> CANARY,

then we would like to tentatively infer (given no information to the contrary) that Tweety flies. So this is a kind of transitivity inference (or simple property inheritance), but one which leads to conclusions 'by default'. Now suppose we also know that

TWEETY —e–> FLEDGLING-CANARY —s–>CANARY,

i.e., Tweety is a fledgling canary and all fledgling canaries are canaries. Further, we know that generally fledgling canaries don't fly, though maybe some particularly precocious ones do. So this is another generic property, but a negative one. We'll notate this by 'crossing out' a generic arc:

FLEDGLING-CANARY –|–> FLY

But now we have a conflict, if we try to answer the question, "*Does Tweety fly?*". One path in the network, running from TWEETY to FLEDGLING-CANARY to CANARY to FLY (the last step, by a default arc), leads to the conclusion that Tweety flies, while the other path, from TWEETY to FLEDGLING-CANARY to the negation of FLY (where the last arc is a negated default arc), leads to the opposite conclusion.

Clearly, if default links were given a universal interpretation (and negative default links a universal negative interpretation), we would have an unsatisfiable network. Even if we do allow for exceptions, it's not clear what conclusion we should draw, given the two paths. Should we perhaps draw no conclusion at all about Tweety's flying status, given the contrary bits of evidence?

Well, intuitively, the conclusion in this case should be that Tweety *doesn't* fly. This conclusion can be assured if we stipulate that *default property inheritance from a lower type node 'wins' over property inheritance from any node lying higher on the same inheritance path*. Intuitively, the information based on fledgling canaries is more specific than the information based more generally on canaries (which lie above fledgling canaries in the type hierarchy); this more specific information should override the more general. It is possible to justify such a stipulation if we use a probabilistic semantics for generic statements,[9] but that's beyond our current purview.

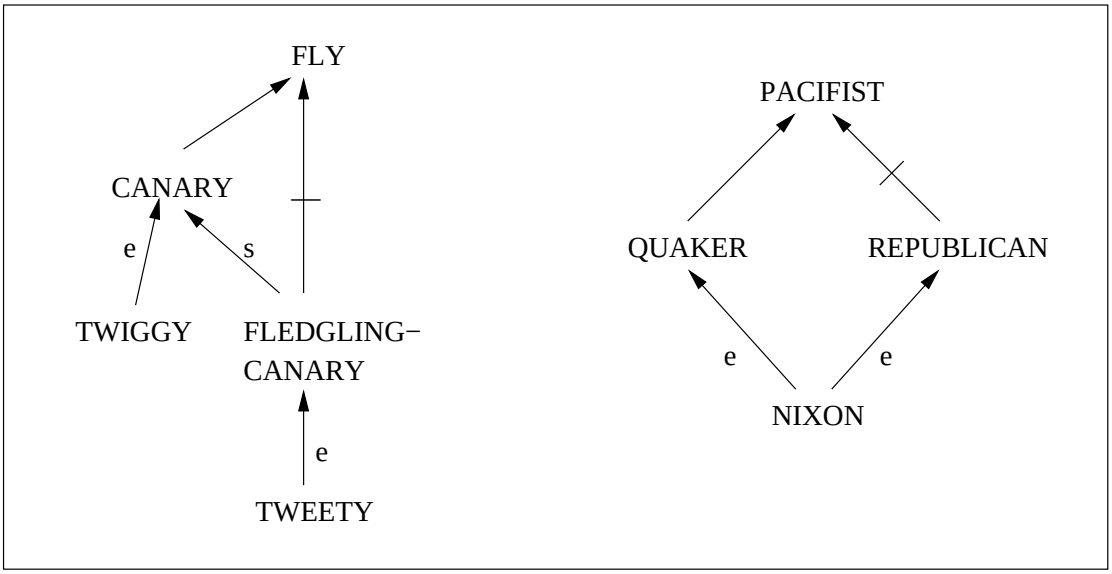
There are cases where we have conflicting evidence, but the above stipulation does not help us arbitrate a conclusion. A famous example is the 'Nixon diamond': we are given

NIXON —e–> QUAKER —->PACIFIST, and

[9]See F. Bacchus, *Representing and Reasoning with Probabilistic Knowledge*, MIT Press, 1990.

NIXON —e–> REPUBLICAN –|–>PACIFIST,

i.e., Nixon is a quaker and quakers are (generally) pacifists; but Nixon is also a Republican, and Republicans are (generally) not pacifists. In the corresponding network there is of course only one node for NIXON and one for PACIFIST, hence the four links form a "diamond". In this case it seems reasonable to abstain from drawing a conclusion – there is simply no way to arbitrate among the conflicting 'arguments' for and against the conclusion.[10]



(a) Does Tweety fly? (b) Is Nixon a pacifist?

Fig. 13: Conflicts in default inheritance

Rules have been invented for more complicated cases, where we have multiple default arguments for and against a conclusion, and the paths corresponding to these arguments overlap in various ways. Some of these rules can be justified by probabilistic semantics, but in general the whole area of default reasoning is not yet on very firm semantic ground.

[10]However, if we had some knowledge of *proportions* of quakers who are pacifists and proportions of Republicans who are nonpacifists, we might be able to draw some reasonable tentative conclusion.