Frames

Frames are "packets" of information about types of entities and their instances, intended to be useful for tasks such as classifying new instances, anticipating their attributes, and inferring the presence and properties of their parts or participants.

Nature and origin of frames

A *frame* is a representation of the most essential facts about some structured entity, with "slots" for its parts, attributes or participating entities, with information about the entity as a whole and about the slots, their default values and their relationships. The entity may be concrete, like a bird, or abstract, like a customer account or a lawsuit. And it may be particular (e.g., a particular bird, Tweety) or general (e.g., the general concept of a canary, or of taking a cab).

As a representation of factual knowledge, frames can be regarded as providing a subset of FOL, but the emphasis is as much on *structuring* knowledge, as on factual content. The following features (except the last) are central to most frame-based representations:

1. Knowledge about an entity and its parts/participants/attributes is structured as a coherent packet (a frame, or a frame system);
2. Parts/participants/attributes are represented as *slots* and *values* (much as we saw in the working memory elements of production systems); e.g., for a bird frame there might be slots for parts such as beak, head, wings, etc., and attributes such as color, food preference, etc. Logically, slots are functions from individuals to individuals (or sometimes sets); e.g., from birds to their beaks or from birds to their color (with colors viewed as individuals). This implies an emphasis on using (unary) *functions* for representing properties, e.g. (deviating slightly from conventions about capitalization we used for FOL),

   color(TWEETY) = YELLOW

   (where YELLOW is an abstract individual) as opposed to

   YELLOW(TWEETY),

   (where YELLOW is a 1-place predicate); or,

   children(FAMILY90210) = set-of(BRANDON-WALSH,BRENDA-WALSH)

   (where *children* is a function from individuals to sets[1]) as opposed to

[1] or sets to sets, if FAMILY90210 is regarded as a set

Child-in(BRANDON-WALSH,FAMILY90210),
Child-in(BRENDA-WALSH,FAMILY90210),

(where *Child-in* is a binary predicate on individuals).

3. Frames often supply *type constraints* and *default values* for slots. For instance, the "beak" slot of a bird frame may be typed as "BIRD-BEAK" (i.e., the value of the beak function is an entity of type BIRD-BEAK, as opposed to, say, a teapot beak); the *children* slot of a FAMILY frame may be typed as PERSON; the default value of the *color* slot of a CANARY frame may be YELLOW, etc.
4. Frames are typically connected in *type hierarchies*, where information about higher-level frames (including their slots and defaults) is assumed to be inherited by the lower-level frames they *subsume*. The most important inference mechanisms for frames are usually *subsumption* and *classification* mechanisms that check whether a given new frame instance is subsumed by a given frame, or determine which frames in a hierarchy subsume a given frame instance (thus classifying it).
5. Multiple interconnected frames may be used represent different "perspectives" on a given type of entity. For instance, we may have connected frames for the different views we have of a building when we walk around it, or move around in it. As we move, we shift from frame to frame within such a *frame system*, where the interconnections join objects that are common to "adjacent" frames. (e.g., the exterior walls and roof of the building may be present in each frame, but in some frames some walls will be marked as visible that are occluded in others.

Frames became popular in AI as a result of the work of several people, including Terry Winograd and Roger Schank (whose notion of a "script" for stereotyped events is closely related); but the single most influential paper was Marvin Minsky's "A framework for representing knowledge" (in P. Winston, ed., *The Psychology of Computer Vision*, McGraw-Hill, 1975). The puzzle that Minsky tried to address was the sheer speed with which people can "orient" themselves to familiar, stereotyped situations, such as walking into a classroom and doing something routine like finding a place to sit, or going to the board to lecture, checking the time on the clock on the wall, etc. He noted that we have strong expectations on what we will find before we even enter the room; we would be most surprised and disoriented if, upon walking through the door, we found ourselves on a sunny beach. His hypothesis was that we have ready-made frame systems at our disposal that provide these expectations about stereotyped situations and objects, so we need not *construct* the entire representation of such an entity from scratch when we encounter it, but only retrieve appropriate frames, and verify and modify aspects of it (e.g., change attribute values that are not as expected).

In Minsky's conception, frames would provide access to both factual and procedural knowledge (e.g., how to switch on the room light), and systems of interconnected frames would provide the shifting perspectives or time-dependent change in a situation, as we move around or as some typical sequence of events unfolds. Actually, the idea of a frame

system (point 5) is the one Minsky thought most important. He cites the psychologist F.C. Bartlett as already having anticipated frames and frame systems in his notion of *schemas*:

> "To serve adequately the demands of a constantly changing environment, we have not only to pick items out of their general setting, but we must know what parts of them may flow and alter without disturbing their general significance and function."

(This is part of a Bartlett quote in M. Minsky, *The Society of Mind*, Simon & Schuster, 1986.) Ironically, the main impact of Minsky's article was to spawn frame formalisms for implementing points 1-4, rather than 5. In that respect, the effect was in a curious way somewhat regressive as well: the initial emphasis in the efforts to implement Minsky's frames was on representing knowledge in terms of slots and values, and this is precisely the kind of representation that had been popular under a slightly different name – *attributes* and values – in the very early days of AI, i.e., the late 50's and early 60's where people built knowledge bases about baseball or military resources using attribute-value representations such as

loc(GAME17) = NY
home-team(GAME17) = DODGERS
visitors(GAME17) = PIRATES
year(GAME17) = 1949
etc.

This is the same type of representation that is used in *relational databases*, which in essence are just large attribute-value tables. Gradually such representations were supplanted in AI by logical ones (or semantic nets) that allow arbitrary predication, logical connectives, and quantification, and do not force properties into a functional format. Eventually, of course, the language of frames was enriched to allow at least limited forms of quantification and connectives, though often with restrictions intended to guarantee that inference processes such as subsumption would remain tractable.

The subsequent development of *description logics* (or *terminological logics*) had its roots at least in part in frames. These typically divide knowledge into "terminological knowledge" and "assertions". The terminological part of a KB defines concepts in frame-like fashion, interrelating the defined concepts in subsumption hierarchies and providing efficient mechanisms for subsumption and classification. The assertional part typically provides some subset of FOL to formulate nondefinitional facts about the application domain. Two well-known examples of description-logic based, implemented (and widely marketed) representations were R. Brachman's CLASSIC (at AT&T)and R. McGregor's LOOM (at ISI); nowadays, OWL-DL (OWL description logic), intended for the semantic web, and other powerful DLs such as ALC (attributive concept language with complements) are more popular.

The usage – and limitations – of frames and description logics will become apparent from the details that follow. Most importantly, they do not provide for expressing

relationships among the parts or participants specified in the "slots" (or "roles") of a frame, even for such simple things as saying that the head of a hammer (in a "hammer frame") is *attached to* the handle, or that the parents in a frame for "family with children" are typically married to each other. We'll therefore be looking at a postscript to these notes pointing out such limitations and suggesting that we need "schemas" that allow greater expressive power to capture the kinds of knowledge people possess.

A syntax for frames

There are many different frame formalisms and description logics; for concreteness we shall pick a somewhat arbitrary, but fairly typical syntax. (The following borrows heavily from notes by James Allen.)

Frame tokens

Reflecting the distinction between *particular* entities and *types* of entities, we distinguish **frame tokens** and **frame types**. Here is an example of a simple frame token representing some attributes of Tweety the canary; alongside it we show the general form of such a frame token:

[TWEETY isa CANARY with	[<token name> isa <type> with
color YELLOW	<slot name> <slot value>
sex MALE	<slot name> <slot value>
age 2	etc.]
health EXCELLENT	
owner DAISY-MILLER]	

So we start with a name (logically, a constant) denoting an individual, followed by the keyword *isa*, followed by a type (logically, a unary predicate), followed by the keyword *with* followed by any number of slot-value pairs (where logically the slots are names of unary functions). So the logical content is equivalent to

CANARY(TWEETY) (or TWEETY $\in$ CANARY, treating CANARY as a set)
color(TWEETY) = YELLOW,
sex(TWEETY) = MALE,
...,
owner(TWEETY) = DAISY-MILLER.

Note that there is some unnaturalness in forcing all of the information into a functional format. Since MALE and EXCELLENT are values of functions, they are regarded as denoting (abstract) individuals; but intuitively they are properties. Furthermore, an attribute like "health" cannot always be specified functionally, but rather can only be *described* using the resources of logic (such as "having an injured wing" or "being

slightly dehydrated"). However, in many practical applications, ones where information is available in a cut-and-dried, database-like format, frames provide a convenient and adequate representation.

We can also represent particular events or situations as frame tokens. For instance, the following represents the event of John eating a specific pizza (PIZZA3) in a particular room at a particular time (it contains essentially the same information as the thematic-role-based semantic net in Fig. 5 of the *Semantic Nets* handout):[2]

[E5 isa EAT with
agent JOHN1
theme PIZZA3
loc LIVING-ROOM1
time T5]

Frame types: using basic term constructors

In the above frame tokens, we supplied explicit values for all slots. However, in general we may not know particular values but only the *types* of those values. For instance, we may know that the theme of the eating event E5 was a pizza, but not *which* pizza it was. So in that case we might use an *indefinite frame type*

(a PIZZA)

to constrain the value of the theme-role. So we might say (instead of the above)

[E5 isa EAT with
agent JOHN1
theme (a PIZZA)
...]

The indefinite "a" in frame type like (a PIZZA) is also called a **term constructor**, since together with a predicate like PIZZA it forms an expression similar to a term (noun phrase) in ordinary language. But we should be careful to distinguish this notion of a "term" from that in FOL, where it meant an individual constant, variable, or functional expression *denoting an individual*. By contrast, a "term" like (a PIZZA) does not denote an individual; rather, it is a predicate, namely PIZZA or, expressing this equivalently as a lambda abstract (in preparation for more complex cases),

[2]Representations of verbs as predicates with an event argument are called *Davidsonian* (after Donald Davidson) or *neo-Davidsonian*, if decomposed using *thematic roles* such as *agent, theme* (the thing acted upon), *instrument*, *location*, etc. A technical difference between thematic roles and slot-filler (attribute-value) representations is that the former are typically viewed as relations, not functions (i.e., not necessarily single-valued or total). So for instance a temporal relation like *before*(E5,NOW34) in Fig. 5 of the *Semantic Nets* handout (i.e., the eating event was in the past relative to a certain time NOW34) does not easily admit a functional representation.

(λ x PIZZA(x)).

Lambda-abstraction is a way of defining a predicate or a function: after the λ header, we specify the independent variable(s) of the predicate or function, and then as the *body* we supply an expression that determines the value of the predicate or function in terms of the values of the independent variable(s). (This is precisely the way we use lambda in Lisp.) When we apply a 1-place lambda-abstract to an argument, we drop the λ and the variable and replace all occurrences of the variable in the body by the given argument. For instance, applying the above lambda abstract to PIZZA3, we obtain

PIZZA(PIZZA3),

i.e., PIZZA3 is a PIZZA. This is called *lambda conversion* (and is the way in which Lisp evaluates nonprimitive functions). Of course the result here is the same as if we had just applied PIZZA to PIZZA3, since (λ x PIZZA(x)) is the same predicate as PIZZA.

So an interpreter for frames needs to distinguish between slot-value pairs like "theme PIZZA3" and "theme (a PIZZA)". From the syntactic form of (a PIZZA), it needs to recognize that this is not an actual value but a type constraint on the value. The meaning of this pair can be expressed logically as

PIZZA(theme(E5)).

Now suppose we also want to specify the agent of E5, JOHN1, to be of type PERSON, but also including the information that the agent's name is John. Further, we want to specify that the pizza was a pepperoni pizza. We can do this by expanding the syntax of indefinite frame types to allow a "with" keyword followed by slot-value pairs. In particular, we can specify the agent and theme as

agent (a PERSON with name "John")
theme (a PIZZA with topping PEPPERONI)

where *name* is a function on individuals and "John" is a string which is the value of that function. (More informatively we might have the string "John Gilmore"; note that an individual's name in English isn't necessarily the same as his name as an internal frame token!). Similarly *topping* is a function whose value in this instance is PEPPERONI. The indefinite types (a PERSON with name "John") and (a PIZZA with topping PEPPERONI) are logically equivalent to the lambda abstracts

(λ x (PERSON(x) $\wedge$ name(x) = "John")), and
(λ x (PIZZA(x) $\wedge$ topping(x) = PEPPERONI)).

The slot specifications say that these predicates hold for agent(E5) and theme(E5) respectively, i.e., the slot specifications are equivalent to

PERSON(agent(E5)) $\wedge$ name(agent(E5)) = "John", and
PIZZA(theme(E5)) $\wedge$ topping(theme(E5)) = PEPPERONI.

Since indefinite types may contain slot-value specifications, we may again want to place

type constraints on those internal slots. For instance, suppose we want to say that John's home has an oil furnace (which might be relevant in an insurance knowledge base); we can express this as

```
[JOHN1 isa PERSON with
            ...
            home (a HOUSE with heating-unit (a OIL-FURNACE))
            ...]
```

Thus we can embed indefinite types within other indefinite types. Logically, the above slot specification is equivalent to

PERSON(JOHN1) $\wedge$
HOUSE(home(JOHN1)) $\wedge$ OIL-FURNACE(heating-unit(home(JOHN1)).

Using indefinite types, we can now specify **defined types** as well. For this we use two types of "definitions", namely partial and full definitions. A partial definition specifies a new type as a subtype of another frame type, providing necessary properties that instances (tokens) of the new type must have (but not sufficient ones in general). The following is a (very!) partial definition of a PERSON:

```
(Define PERSON inherits-from
        (a ANIMAL with name (a STRING)
                       home-address (a ADDRESS)
                       age (a NUMBER)))
```

Logically, *inherits-from* corresponds to implication; i.e., the above partial definition is equivalent to

$$\begin{aligned}\forall x.\ \text{PERSON}(x) \Rightarrow\ & \text{ANIMAL}(x) \wedge \\ & \text{STRING}(\text{name}(x)) \wedge \\ & \text{ADDRESS}(\text{home-address}(x)) \wedge \\ & \text{NUMBER}(\text{age}(x))\end{aligned}$$

By saying that PERSON inherits-from ANIMAL, all the slots and values that are associated with ANIMAL also apply to PERSON, in addition to the ones explicitly specified for PERSON.

A *full* definition also defines a subtype of another frame type, and supplies *necessary and sufficient* conditions for an instance (token) to belong to that subtype. For instance, the following is a full definition of a BACHELOR:

```
(Define BACHELOR as
        (a PERSON with sex MALE
                       marital-status SINGLE))
```

Note the use of the keyword *as* instead of *inherits-from* here. The logical representation in this case involves *equivalence*:

$$\forall x.\ \text{BACHELOR}(x) \Leftrightarrow \text{PERSON}(x) \wedge \text{sex}(x) = \text{MALE} \wedge \text{marital-status}(x) = \text{SINGLE}.$$

Thus any token that is a single, male PERSON is guaranteed to be a BACHELOR (not just the other way around). In the case of the partial definition of PERSON, there was no assurance that an ANIMAL with a name, address, and age was a PERSON (perhaps it is a household pet).

Term constructors for sets

Other term constructors besides "a" have been found very useful in practice. One is "a-set", allowing us to construct terms constraining sets rather than individuals. These are of form

(a-set (*term expression*) with ...)

where the slot-value pairs after "with" may specify attributes of the set, most often its cardinality. E.g.,

(a-set (a COURSE) with cardinality 2)

specifies a set of size 2 in which each member is (a COURSE), i.e., this is a predicate that holds for any two courses; logically, it is

$$\lambda\ s.\ (\forall x.\ x \in s \Rightarrow \text{COURSE}(x)) \wedge \text{cardinality}(s) = 2.$$

As a slight elaboration of the example, if we want to specify a pair of AI courses taught by Minsky, we might use

(a-set (a COURSE with subject AI instructor MINSKY) cardinality 2).

This would elaborate the previous constraint on each element, COURSE(x), to

$$\text{COURSE}(x) \wedge \text{subject}(x) = \text{AI} \wedge \text{instructor}(x) = \text{MINSKY}.$$

Another useful constructor for set description is "one-of", which allows us to specify the members of a set explicitly. For example,

(one-of {WHITE, YELLOW, GREEN})

is equivalent to a predicate

$$\lambda\ x.\ x = \text{WHITE} \vee x = \text{YELLOW} \vee x = \text{GREEN}.$$

We could use this to specify that Tweety (or more generally, a canary) is white, yellow or green, using (one-of {WHITE, YELLOW, GREEN}) as a type constraint on the *color* slot. We are now also able to represent who the children of FAMILY90210 are:

[FAMILY90210 isa FAMILY with
children (a-set (one-of BRANDON-WALSH, BRENDA-WALSH)
with cardinality 2) ...]

Thus we are specifying that the value of children(FAMILY90210) is a set each of whose members is either BRANDON-WALSH or BRENDA-WALSH. It is not redundant to specify the cardinality as 2, since it may be that the two names are names for the same individual. (However, under a *unique names* assumption, i.e., an assumption that all names denote distinct individuals, the cardinality attribute would be redundant.)

Default values

Finally, we need a way of specifying *default values* of slots, which after all was a key feature in Minsky's conception of frames. We do this with a special slot, *default*, that doesn't have an ordinary functional interpretation.

For example, we might express that canaries are generally yellow (though they might occasionally be, say, white or pale green) by specifying a color slot in the (partial) definition of a canary as

color (a COLOR with default YELLOW)

Similarly we might define a car to have 4 doors by default,

no.-of-doors (a NUMBER with default 4);

or better, we might specify a *set* of doors:

doors (a-set DOOR with cardinality (a NUMBER with default 4)).

As a more complete example, suppose that a Rochester travel agent wants to set up "booking records" for flights reserved by customers. In the process of supplying the agent with the required data, a customer might well say, '*I'd like a flight to Chicago on November 8*' without mentioning the city of origin. In such a case, it would be reasonable to assume that the city of origin is Rochester – unless and until the customer says otherwise. This could be handled by a default origin as shown in the following general frame defining a BOOKING-RECORD:

(Define BOOKING-RECORD as
(a RECORD with
flight-date (a DATE)
origin (a CITY with *default* ROCHESTER)
dest (a CITY)
airline (a AIRLINE)
fare (a NUMBER))

Let's try to write down as before what this means logically:

$\forall$x. BOOKING-RECORD(X) $\Leftrightarrow$

RECORD(x) ∧
DATE(flight-date(x)) ∧
CITY(origin(x)) ∧ *default*(origin(x)) = ROCHESTER
... etc.

Now while there is nothing "illegal" about the functional notation *default*(origin(x)) = ROCHESTER, we want to use that notation in a way that is not valid in FOL. In particular, we want to change this to say simply origin(...) = ROCHESTER when creating a frame token for a particular flight, provided that no explicit origin for the flight was specified. This is a nonmonotonic (rather than deductive) inference, i.e., one subject to retraction in the light of further information.

As mentioned before, frame languages evolved into *description logics* such as CLASSIC and LOOM, which emphasize *subsumption* and *classification* of complex types as their primary inference method. An example of subsumption might be determining whether *a person with age > 40 and with a child who is a student* subsumes *a man with age > 50 and with a daughter who is a college student and a son who is a programmer*. Here "subsumes" means "is more general than". (One can imagine that such questions might be of interest in a Census Bureau application, for example.) Similarly classification tries to find those types known to the terminological knowledge base that directly subsume a given description. Early in this development, it was generally argued that the expressive capabilities of the definitional language must be limited in such a way that subsumption will be tractable – e.g., low-order polynomial-time in the size of the query and KB. However, practical applications exerted a push towards greater expressiveness, and current description logics tend to be exponential-time in the worst case (though not typically in the "average" case!), and have even moved toward undecidability.

Procedural attachment

As mentioned earlier, Minsky viewed frames as packets containing both declarative and procedural knowledge. The Brachman & Levesque text puts particular emphasis on the procedural aspect, allowing values of slots to specify procedures. There are two types of these procedures, **IF-NEEDED** and **IF-ADDED**. **IF-NEEDED** routines compute the value of a slot, much as in production systems, except that procedures within a frame are allowed to reference not only parameters of that frame, but also parameters of other, related frames. They do this using successions of slot names, much as we access fields of record structures in various programming languages. **IF-ADDED** routines, rather than computing the value of the slot they are specified for, are triggered when a value for the slot is *supplied*, and may fill in values of other slots, within the given frame instance or related frame instances, or even create new frame instances.

For example, B & L develop and example of air travel planning, and one of their proposed frames (for one leg of a trip) has both :PreviousStep and :NextStep slots (these are nil for the first and final leg respectively). When the :PreviousStep of an instance of this frame is supplied by the user, an **IF-ADDED** routine is triggered that accesses

:PreviousStep:NextStep (i.e., the :NextStep slot of the previous step) and, if this has no value yet, sets it to the current step.[3] A similar **IF-ADDED** routine could be attached to the :NextStep slot. This ensures that two successive steps are connected in both directions. As well, if a :NextStep value is added, another **IF-ADDED** procedure is triggered that checks whether the two successive legs of the trip connect on the same day or not, and if not, a LodgingStay frame instance is created, whose date and default location are determined by the date and city where the connection is made.

B & L make an analogy between this kind of constraint maintenance and spreadsheets, and mention that frames can be effectively used for maintaining the integrity of relational databases. However, it seems that a less risky, more formal approach would be to employ declarative integrity constraints that are systematically enforced by some interpretive system. More generally, a problem with both production systems and frame-based systems is that the proceduralization of inference within the domain is not constrained by any formal semantic notion of entailment, and thus risks error, inconsistency, and opacity.

In the view of this writer, frames (and the similar, more dynamically oriented *scripts* of Roger Schank) are important ways of "packaging together" knowledge items pertaining to related entities, such as the parts of a type of entity and their properties, relations, and functions; or such as the prototypical subevents in familiar sequences of events, like those comprising "dining at a restaurant", or "brushing your teeth". However, it seems ultimately counterproductive to limit the logical expressions allowed in these structures to simple attribute values and type predications, preventing us from easily and fully "saying" what we know about them. The right way forward seems to be to allow full language-like expressivity within frames. Current research in our Department is concerned with using fully expressive *schemas* as a very general way to package frame-like and script-like knowledge, and a preliminary implementation in a "virtual human" (albeit, as yet without significant inferential capabilities) has proved useful for controlling dialogue.

Further References

H. Levesque, *A fundamental tradeoff in knowledge representation and reasoning.*
P.J. Hayes, *The logic of frames*, in D. Metzing (ed.), *Frame Conceptions and Text Understanding*, New York: de Gruyter, 1979.

[3]The example indicates why B & L use leading colons on their slot names, i.e., to be able to concatenate them unambiguously in specifying slot access paths