Knowledge Representation & Reasoning

Shyamanta M Hazarika
CSE, SoE
Tezpur University

Reasoning with frames

Basic (local) reasoning goes like this:

1. user instantiates a frame, i.e., declares that an object or situation exists
2. slot fillers are inherited where possible
3. inherited **IF-ADDED** procedures are run, causing more frames to be instantiated and slots to be filled.

If the user or any procedure requires the filler of a slot then:

1. if there is a filler, it is used
2. otherwise, an inherited **IF-NEEDED** procedure is run, potentially causing additional actions

Globally:

- make frames be major situations or object-types you need to flesh out
- express constraints between slots as **IF-NEEDED** and **IF-ADDED** procedures
- fill in default values when known

$\Rightarrow$ like a fancy, semi-symbolic spreadsheet

Planning a trip

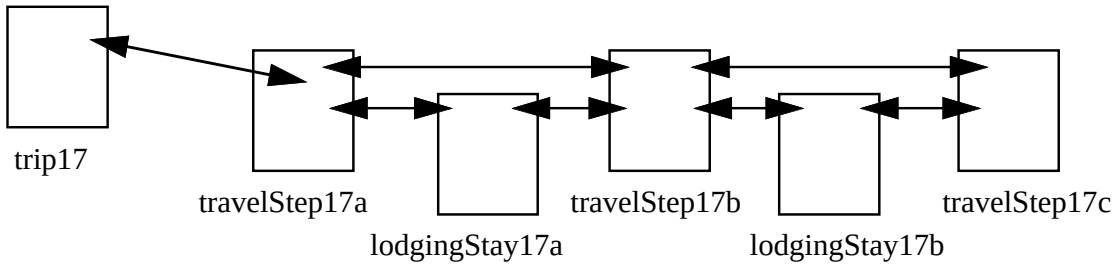
A simple example: a frame system to assist in travel planning (and possibly documentation – automatically generate forms)

Basic structure (main frame types):

- a Trip will be a sequence of TravelSteps

  these will be linked together by slots

- a TravelStep will usually terminate in a LodgingStay (except the last, or one with two travels on one day)
  - a LodgingStay will point to its arriving TravelStep and departing TravelStep
  - TravelSteps will indicate the LodgingStays of their origin and destination



(trip17
<:**INSTANCE-OF** Trip>
<:FirstStep travelStep17a>
<:Traveler ronB> ...)

Parts of a trip

TravelSteps and LodgingStays share some properties (e.g., :BeginDate, :EndDate, :Cost, :PaymentMethod), so we might create a more general category as the parent frame for both of them:

(Trip
<:FirstStep TravelStep>
<:Traveler Person>
<:BeginDate Date>
<:TotalCost Price> ...)

(TripPart
<:BeginDate>
<:EndDate>
<:Cost>
<:PaymentMethod> …)

(TravelStep
<:**IS-A** TripPart>
<:Means>
<:Origin> <:Destination>
<:NextStep> <:PreviousStep>
<:DepartureTime> <:ArrivalTime>
<:OriginLodgingStay>
<:DestinationLodgingStay> …)

(LodgingStay
<:**IS-A** TripPart>
<:ArrivingTravelStep>
<:DepartingTravelStep>
<:City>
<:LodgingPlace> …)

Travel defaults and procedures

Embellish frames with defaults and procedures

```
(TravelStep
     <:Means  airplane> ...)

(TripPart
     <:PaymentMethod  visaCard> ...)

(TravelStep
     <:Origin [IF-NEEDED {if no SELF:PreviousStep then newark}]>)
(Trip
     <:TotalCost
        [IF-NEEDED
          { x←SELF:FirstStep;
           result←0;
           repeat
            {  if  exists x:NextStep
               then
                    { result←result + x:Cost +
                              x:DestinationLodgingStay:Cost;
                      x←x:NextStep }
             else  return  result+x:Cost }}]>)
```

Program notation (for an imaginary language):

- SELF is the current frame being processed
- if *x* refers to an individual frame, and *y* to a slot, then *xy* refers to the filler of the slot

assume this is 0 if there is no LodgingStay

More attached procedures

```
(TravelStep
      <:NextStep
         [IF-ADDED
           {if  SELF:EndDate ≠ SELF:NextStep:BeginDate
            then
                SELF:DestinationLodgingStay ←
                     SELF:NextStep:OriginLodgingStay ←
                          create new LodgingStay
                               with :BeginDate = SELF:EndDate
                               and with :EndDate = SELF:NextStep:BeginDate
                               and with :ArrivingTravelStep = SELF
                               and with :DepartingTravelStep = SELF:NextStep
                   …}]>
  …)
```

Note: default :City of LodgingStay, etc. can also be calculated:

```
(LodgingStay
      <:City [IF-NEEDED  {SELF:ArrivingTravelStep:Destination}]…>   ...)
```

Frames in action

Propose a trip to Toronto on Dec. 21, returning Dec. 22

(trip18
<:**INSTANCE-OF** Trip>
<:FirstStep travelStep18a>)

(travelStep18a
<:**INSTANCE-OF** TravelStep>
<:BeginDate 12/21/98>
<:EndDate 12/21/98>
<:Means>
<:Origin>
<:Destination toronto>
<:NextStep> <:PreviousStep>
<:DepartureTime> <:ArrivalTime>)

the first thing to do is to create
the trip and the first step

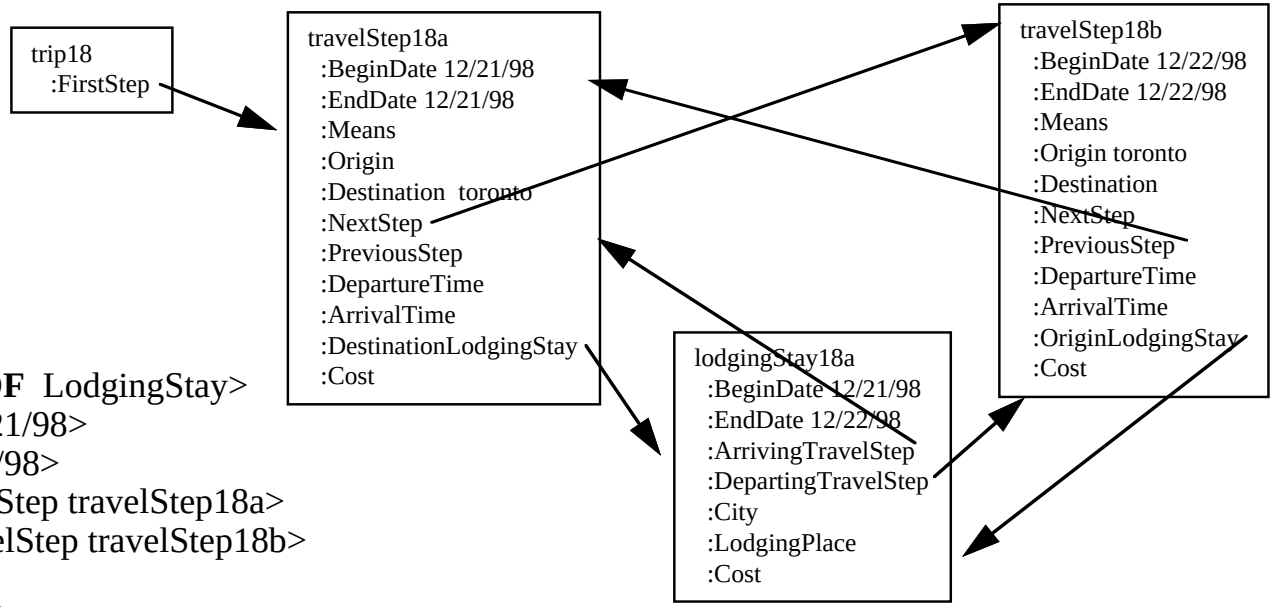
(travelStep18b
<:**INSTANCE-OF** TravelStep>
<:BeginDate 12/22/98>
<:EndDate 12/22/98>
<:Means>
<:Origin toronto>
<:Destination>
<:NextStep>
<:PreviousStep travelStep18a>
<:DepartureTime> <:ArrivalTime>)

the next thing to do is to create
the second step and link it to the first
by changing the :NextStep

*(*travelStep18a
<:NextStep travelStep18b>*)*

Triggering procedures

IF-ADDED on :NextStep then creates a LodgingStay:



(lodgingStay18a
<:**INSTANCE-OF** LodgingStay>
<:BeginDate 12/21/98>
<:EndDate 12/22/98>
<:ArrivingTravelStep travelStep18a>
<:DepartingTravelStep travelStep18b>
<:City>
<:LodgingPlace>)

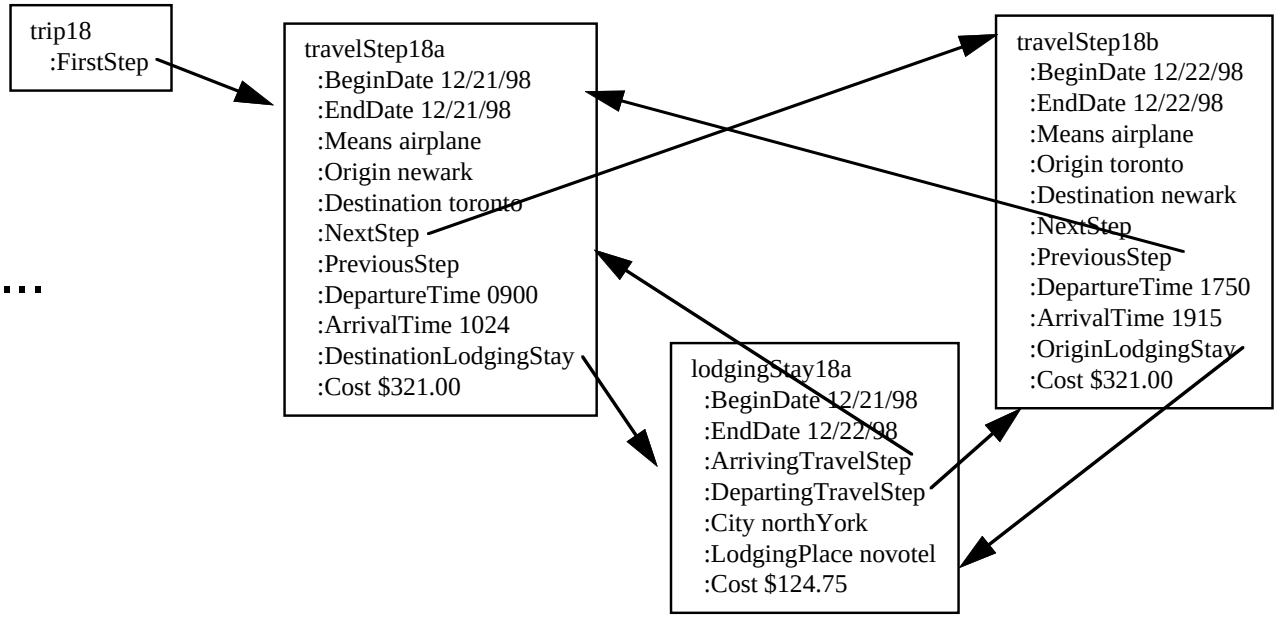
If requested, **IF-NEEDED** can provide :City for lodgingStay18a (toronto)

which could then be overridden by hand, if necessary
(e.g. usually stay in North York, not Toronto)

Similarly, apply default for :Means and default calc for :Origin

Finding the cost of the trip

So far...



Finally, we can use :TotalCost **IF-NEEDED** procedure (see above) to calculate the total cost of the trip:

- result← 0, x←travelStep18a, x:NextStep=travelStep18b
- result←0+$321.00+$124.75; x← travelStep18b, x:NextStep=NIL
- return: result=$445.75+$321.00 = $766.75

Using the formalism

Main purpose of the above: embellish a sketchy description with defaults, implied values

- maintain consistency
- use computed values to
  1. allow derived properties to look explicit
  2. avoid up front, potentially unneeded computation

Monitoring

- hook to a DB, watch for changes in values
- like an ES somewhat, but monitors are more object-centered, inherited

Scripts for story understanding

generate expectations (e.g., restaurant)

Real, Minsky-like commonsense reasoning

- local cues $\Rightarrow$ potentially relevant frames $\Rightarrow$ further expectations
- look to match expectations ; mismatch $\Rightarrow$ “differential diagnosis”

Extensions

1. Types of procedures
   - **IF-REMOVED**

     e.g., remove TravelStep ⇒ remove LodgingStay
   - "servants" and "demons"

     flexible "pushing" and "pulling" of data
2. Slots
   - multiple fillers
   - "facets" – more than just defaults and fillers
     - [**REQUIRE** <class>] (or procedure)
     - **PREFER** – useful if conflicting fillers
3. Metaframes

   (CanadianCity <:**INSTANCE-OF** GeographicalCityType> …)

   (GeographicalCityType <:**IS-A** CityType>
   <:AveragePopulation NonNegativeNumber> …)
4. Frames as actions ("scripts")

Object-oriented programming

Somewhat in the manner of production systems, specifying problems with frames can easily slide into a style of *programming*, rather than a declarative object-oriented modeling of the world

- note that direction of procedures (pushing/pulling) is explicitly specified

  not declarative

This drifts close to conventional object-oriented programming (developed concurrently).

- same advantages:
  - definition by specialization
  - localization of control
  - encapsulation
  - etc.
- main difference:
  - frames: centralized, conventional control regime (instantiate/ inherit/trigger)
  - object-oriented programming: objects acting as small, independent agents sending each other messages