Frames – object-oriented representation

FOL and production systems are representation methods that are “flat”, in the sense that each representation unit is independent of the others.

Knowledge about an object could be scattered all over the knowledge base, among unrelated sentences. In big KBs, this can become critical, therefore it is important to organize knowledge representation somehow.

In 1975, Marvin Minsky introduced “frames” as knowledge representation for object-oriented groups of procedures to *recognize and deal with new situations*.

In representations using frames, facts and rules are grouped in terms of the kind of objects they belong to. Thus, knowledge is not seen as just a collection of sentences, but rather it is structured in terms of what knowledge is about (i.e., the objects of knowledge).

Frames – object-oriented representation

Generic and individual frames

Individual frames represent objects; generic frames represent categories of classes of objects.

Individual frames are similar to WMEs in production systems. An individual frame is a named list of slots into which values, called fillers, can be dropped:

(Frame_name
<slot_ name1 filler1>
<slot_name2 filler2>
…)

The names of individual frames begin with a lower-case; the names of generic frames begin with upper-case.

Fillers are atomic values (numbers or strings) or names of other individual frames or generic frames.

Slot names begin with upper-case and are prefixed with :

Frames – object-oriented representation

Examples

(CanadianCity
<:IS-A City>
<:Province CanadianProvince>
<:Country canada>)

(toronto
<:INSTANCE-OF CanadianCity>
<:Province ontario>
<:Population 4.5M>…)

Frames – object-oriented representation

Examples

(CanadianCity
<:IS-A City>
<:Province CanadianProvince>
<:Country canada>)

(toronto
<:INSTANCE-OF CanadianCity>
<:Province ontario>
<:Population 4.5M>…)

Individual frames have a special slot called :INSTANCE-OF, whose filler is the name of a generic frame which indicates the category of the object represented by that individual frame (the individual frame is an instance of the generic frame).

Generic frames have a special slot called :IS-A, whose filler is the name of a more general generic frame. We say that the generic frame is a specialization of the more general one.

Frames – object-oriented representation

Slots of generic frames can have attached procedures **IF-ADDED** and **IF-NEEDED**.

The rule Parent(x,y) ⇐ Mother(x,y) can be procedurally interpreted as following:

IF-NEEDED – whenever we have to solve a goal that matches Parent(x,y), we reduce it to solving Mother(x,y) (backward chaining). Procedurally, the connection between mothers and parents is made when we must prove something about parents.

IF-ADDED – whenever a fact that matches Mother(x,y) is added to the KB, we also add Parent(x,y) (forward chaining). The connection between mothers and parents is made when we know something new about Mother(x,y).

Frames – object-oriented representation

(Trip

<:TotalCost [IF-NEEDED ComputeCost]>…)

(Lecture

<:DayOfWeek WeekDay>

<:Date [IF-ADDED ComputeDayofWeek]>…)

A slot can have both a filler and an (inherited) attached procedure in the same frame.

Frames – object-oriented representation

Inheritance

In a frame system, the reasoning involves creating individual instances of generic frames and filling/inferring values into slots. Generic frames can be used to fill in values that are not explicitly given at the creation of the instances. Generic frames can also trigger additional actions when fillers are provided. We can determine a value in an instance by inheritance (the child frames inherit properties from parent frame).

For example, if we are interested in the filler for the slot :Country of the toronto frame, we can use :INSTANCE-OF that indicates to the generic frame CanadianCity, where the value is given (toronto inherits the :Country property from CanadianCity).

If toronto had not a filler for :Province, we should still know by inheritance that we should look for an instance of CanadianProvince.

(CanadianCity
<:IS-A City>
<:Province CanadianProvince>
<:Country canada>)

(toronto
<:INSTANCE-OF CanadianCity>
<:Province ontario>
<:Population 4.5M>…)

Frames – object-oriented representation

Inheritance of attached procedures

(Trip

<:TotalCost [IF-NEEDED ComputeCost]>…)

(ExpensiveTrip

<:IS-A Trip>…)

(ExoticExpensiveTrip

<:IS-A ExpensiveTrip >…)

If we create an instance of the frame Trip and we want to find a filler for :TotalCost in this instance, we use the attached procedure IF-NEEDED.

The same procedure can be used through inheritance if we create an instance of ExoticExpensiveTrip.

Frames – object-oriented representation

Inheritance of attached procedures

If we create an instance of the frame Lecture with the data specified explicitly:

(Lecture
<:DayOfWeek WeekDay>
<:Date [IF-ADDED ComputeDayofWeek]>…)

(lecture1
<:INSTANCE-OF Lecture>
<:Date 20Nov>…)

Then the attached IF-ADDED procedure would be executed, filling the slot :DayOfWeek.

If we later change the :Date slot, the procedure will execute again, changing the filler for :DayOfWeek.

Frames – object-oriented representation

In a frame system, we use an inherited value only if we cannot find a filler otherwise (it is defeasible).

A slot filler in a generic frame can be overridden explicitly in its instances and its specializations:

(Elephant
<:IS-A Mammal>
<:EarSize large>
<:Color gray>…)

(raja
<INSTANCE-OF Elephant>
<EarSize small>…)

(RoyalElephant
<IS_A Elephant>
<:Color white>…)

(clyde
<:INSTANCE-OF RoyalElephant>…)

A frame system allows for multiple inheritance. An individual/generic frame can be an instance/a specialization of more than one generic frame.

(AfricanElephant
<:IS-A Elephant>
<:IS-A AfricanAnimal>…)

[for more on Inheritance, see Chapter 10 in Brachman&Levesque]

Frames – object-oriented representation

Reasoning with frames

In a frame system, reasoning starts when the system recognizes an object being an instance of a generic frame and applies the procedures triggered by that instance. These procedures can produce new data or changes in the KB. The system operation stops when no more procedures are applicable.

The system operates in a three-step loop:

1. Someone (a user or an external system) declares that an object exists, instantiating a generic frame.
2. Any slot fillers that are not provided explicitly, but can be inherited by the new instance, are inherited.
3. For each slot with a filler, if an IF-ADDED procedure can be inherited, then it is executed. By its execution, new slots can be filled or new frames can be instantiated; then go to 1.

Frames – object-oriented representation

If a filler is requested by a user, an external system or an attached procedure, then:

1. If the filler exists, then the value is returned;
2. Otherwise, any IF-NEEDED procedure that can be inherited is executed, computing the filler. The execution can also fill other slots or instantiate new frames.

If no result is produced by the steps above, then the value of the slot is unknown.

Obs. The inheritance of the values is done when the individual frame is created, but IF-NEEDED procedures are invoked only by request.

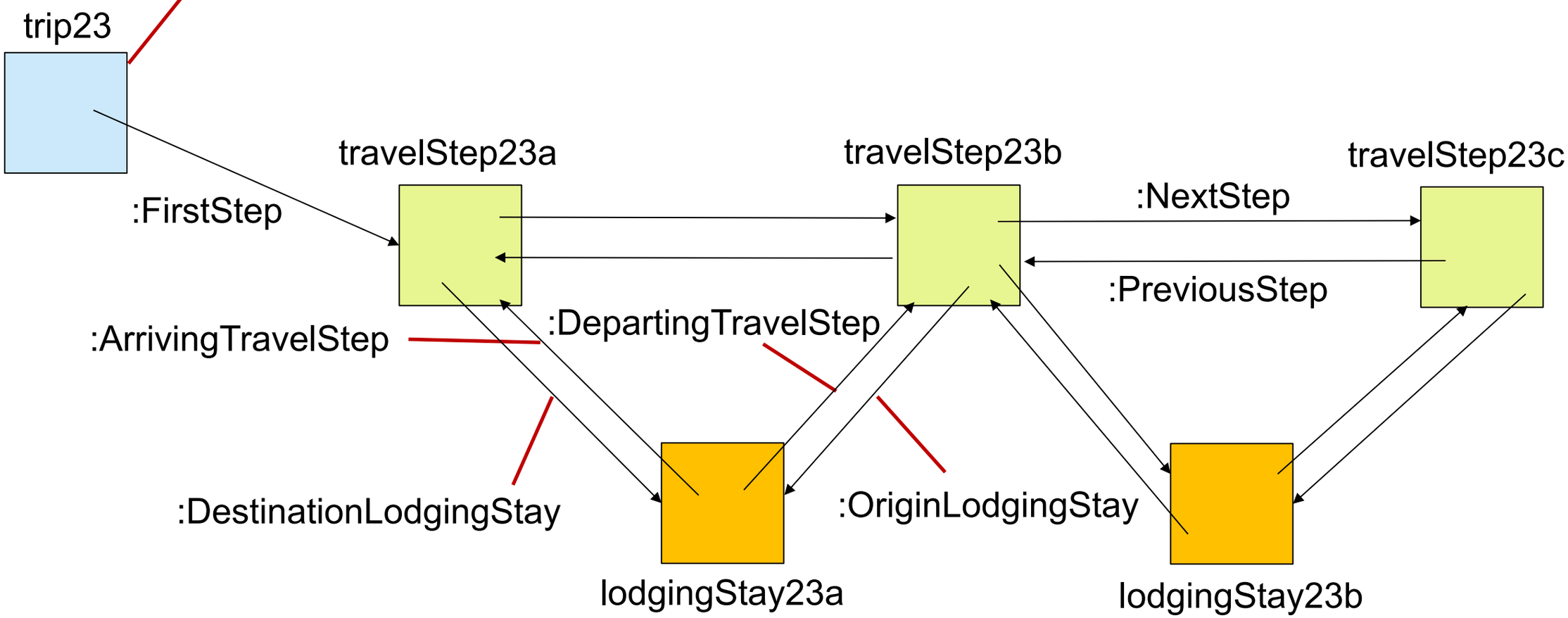
The constraints between slots are expressed by the attached IF-ADDED and IF-NEEDED procedures. The programmer decides whether reasoning is data-directed of goal-directed.

Frames – an example

(trip23
<:INSTANCE-OF Trip>
<:FirstStep travelStep23a>
<:Traveler davidF>
<:BeginDate 23/11/2025>
<:EndDate 25/11/2025>
<:TotalCost 3400RON>)

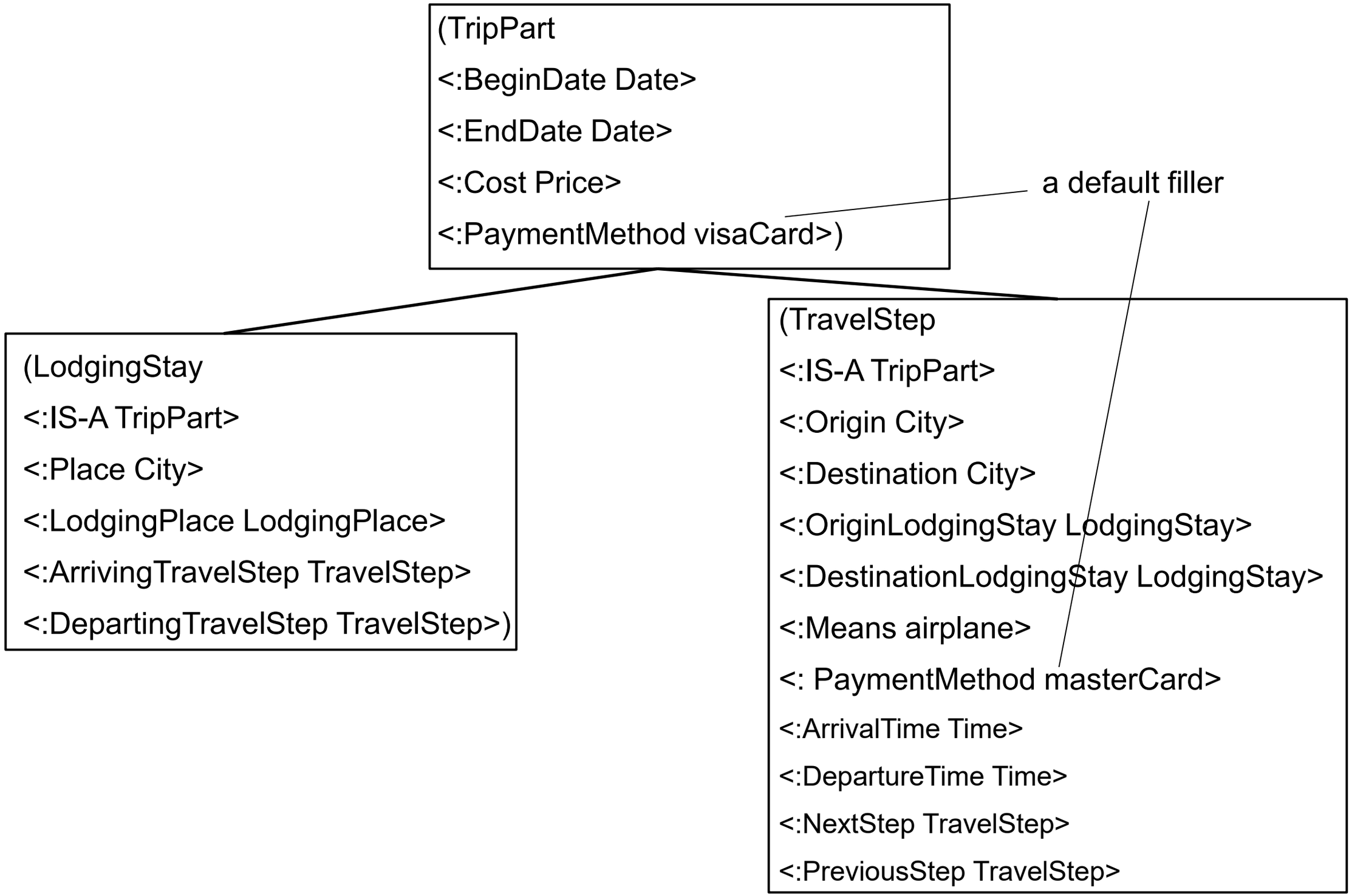
The generic frame

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



Frames – an example

The frames TravelStep and LodgingStay share some properties that we group in the more general frame TripPart



Frames – an example

Notations: if x is an individual frame and y is a slot, then xy refers to the filler of the slot in that frame;

SELF is a reference to the current frame.

```
(TravelStep
<:Origin [IF-NEEDED
      {if no SELF:PreviousStep then otopeni;
        else SELF:PreviousStep:Destination;
}]>…
)
```

```
(TravelStep
<:PreviousStep [IF-ADDED
    {SELF:PreviousStep:NextStep←SELF;}
]>…)
```

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

Frames – an example

```
(Trip
<:TotalCost [IF-NEEDED
    {result←0
     x←SELF:FirstStep;
   ┌ repeat
   │     if exists x:NextStep then
   │             {result←result+c:Cost;
   │              if exists x:DestinationLodgingStay then
   │                   {result←result+x:DestinationLodgingStay:Cost;}
   │              x←x:NextStep;
   │              }
   └       else return result+x:Cost;
    }
]>…)
```

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

Frames – an example

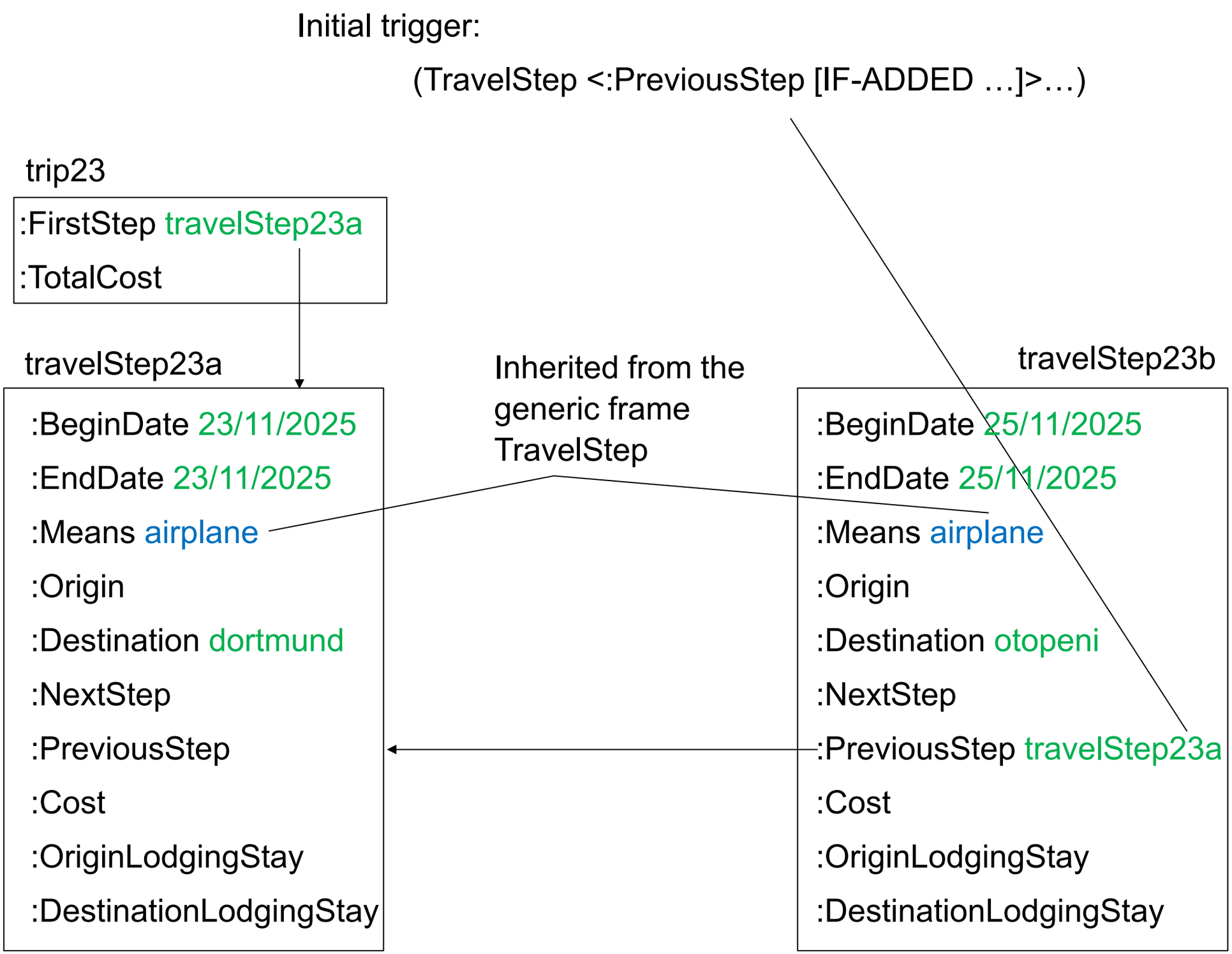
For a certain trip, called trip23, we create an instance of Trip and two instances of TravelStep:

```
(trip23
<:INSTANCE-OF Trip>
<:FirstStep travelStep23a>
)
```

```
(travelStep23a
<:INSTANCE-OF TravelStep>
<:Destination dortmund>
<:BeginDate 23/11/2025>
<:EndDate 23/11/2025>
)
```

```
(travelStep23b
<:INSTANCE-OF TravelStep>
<:Destination otopeni>
<:BeginDate 25/11/2025>
<:EndDate 25/11/2025>
<:PreviousStep travelStep23a>
)
```

Frames – an example



Frames – an example

trip23

:FirstStep travelStep23a

:TotalCost

travelStep23a

:BeginDate 23/11/2025

:EndDate 23/11/2025

:Means airplane

:Origin

:Destination dortmund

:NextStep travelStep23b

:PreviousStep

:Cost

:OriginLodgingStay

:DestinationLodgingStay

travelStep23b

:BeginDate 25/11/2025

:EndDate 25/11/2025

:Means airplane

:Origin

:Destination otopeni

:NextStep

:PreviousStep travelStep23a

:Cost

:OriginLodgingStay

:DestinationLodgingStay

Filler computed by the IF-ADDED procedure (TravelStep <:PreviousStep [IF-ADDED …]>…),

which is triggered by the filler travelStep23a

Frames – an example

trip23

:FirstStep travelStep23a

:TotalCost

travelStep23a

:BeginDate 23/11/2025

:EndDate 23/11/2025

:Means airplane

:Origin

:Destination dortmund

:NextStep travelStep23b

:PreviousStep

:Cost

:OriginLodgingStay

:DestinationLodgingStay lodgingStay23a

travelStep23b

:BeginDate 25/11/2025

:EndDate 25/11/2025

:Means airplane

:Origin

:Destination otopeni

:NextStep

:PreviousStep travelStep23a

:Cost

:OriginLodgingStay lodgingStay23a

:DestinationLodgingStay

:ArrivingTravelStep travelStep23a

:DepartingTravelStep travelStep23b

:BeginDate 23/11/2025

:EndDate 25/11/2025

:Place

:Cost

lodgingStay23a

The instance lodgingStay23a is created by the IF-ADDED procedure inherited by :NextStep in travelStep23a, triggered by the filler travelStep23b

Frames – an example

trip23

:FirstStep travelStep23a

:TotalCost

travelStep23a

:BeginDate 23/11/2025

:EndDate 23/11/2025

:Means airplane

:Origin otopeni

:Destination dortmund

:NextStep travelStep23b

:PreviousStep

:Cost

:OriginLodgingStay

:DestinationLodgingStay lodgingStay23a

travelStep23b

:BeginDate 25/11/2025

:EndDate 25/11/2025

:Means airplane

:Origin dortmund

:Destination otopeni

:NextStep

:PreviousStep travelStep23a

:Cost

:OriginLodgingStay lodgingStay23a

:DestinationLodgingStay

Fillers computed by (TravelStep <:Origin [IF-NEEDED…]>…), if "someone" asks for them

Filler computed by (LodgingStay <:Place [IF-NEEDED…]>…), if "someone" asks for it

lodgingStay23a

:ArrivingTravelStep travelStep23a

:DepartingTravelStep travelStep23b

:BeginDate 23/11/2025

:EndDate 25/11/2025

:Place dortmund

:Cost

Frames – summary

Good candidates for representation by frames are applications that have, in general, common stereotype structures that can be filled with details of particular situations.

Facts are clustered around objects.

Slots contain information of various types: simple values, references to other frames or procedures.

Unfilled slots can be filled through inference.

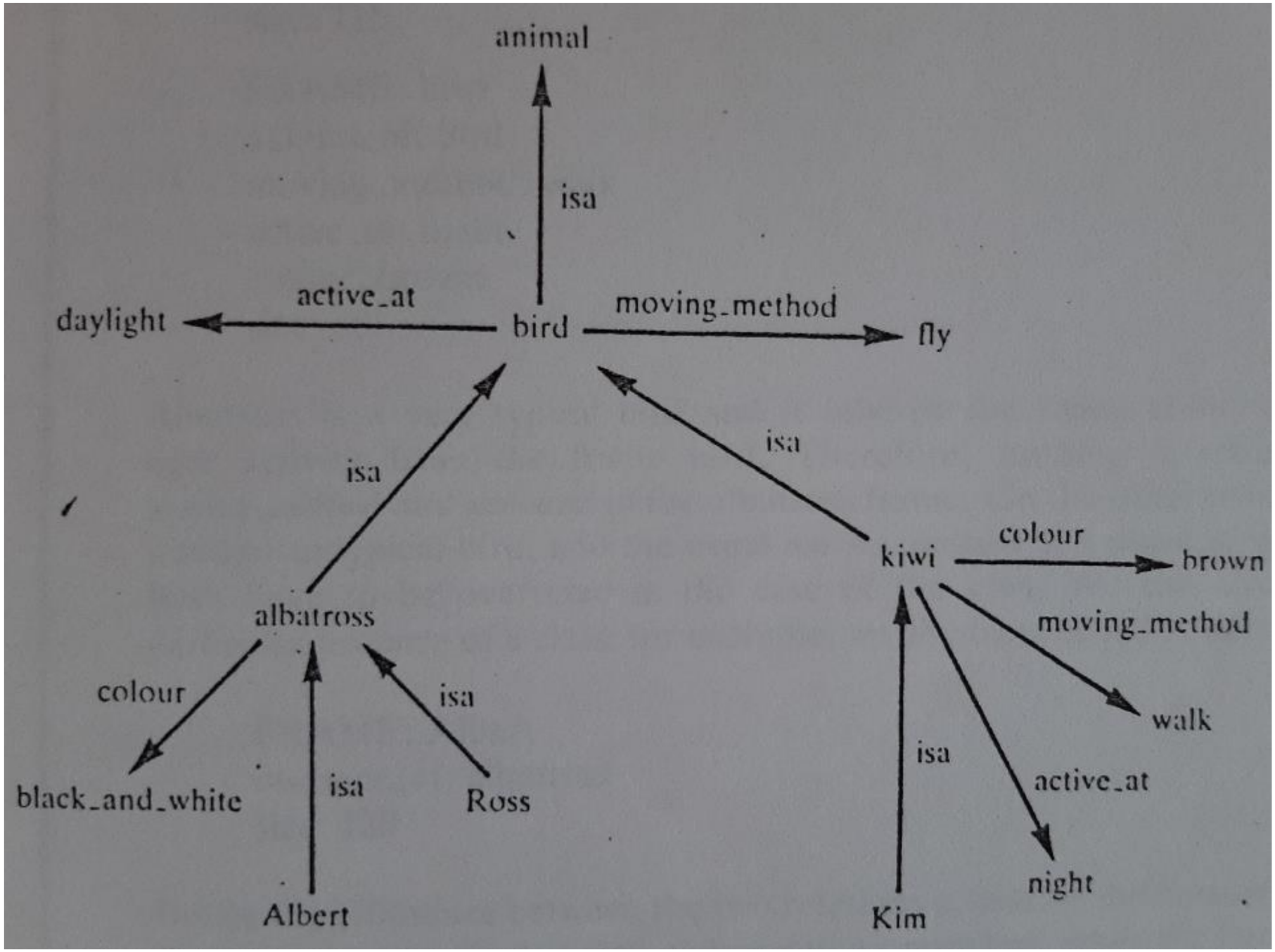
Frame languages have a significant overlap with object-oriented programming (OOP) languages.

The main difference:

- Frame systems have a centralized, conventional control regime that works in a cycle: instantiate a frame, declare some slot fillers, inherit values, trigger procedures and then wait for the next input (i.e., instantiated frame)
- OOP systems tend to be more decentralized, where objects act as independent agents sending each other messages

Frames and semantic networks

A representation of frames in PROLOG can be found in the book of Ivan Bratko, p. 351-355 (2nd edition).



[A semantic network - Figure taken from p. 349]

Frames and semantic networks

https://www.linkedin.com/pulse/wordnet-word-sense-disambiguation-wsd-nltk-aswathi-nambiar/

https://www.professional-ai.com/difference-between-semantic-net-and-frame.html

What Is the Difference Between Semantic Net and Frame?

Semantic Net:
A semantic network is a representation of knowledge in the form of graphs, using real-world meanings that are easy to understand (e.g., "is a" and "is a part“ for inheritance hierarchies). It's a widely popular idea in artificial intelligence and natural language processing because it supports reasoning. The core idea of the semantic network is that knowledge can be stored in graphs where nodes represent objects in the world, and the arcs show the relationships between objects.

Frame:
The frame is also a data structure to represent similar properties and possibilities for knowledge representation like the semantic networks. It contains the same ideas of inheritance and default values. However, frames are more powerful than the semantic network because their slots contain instructions to be understood well for computing things stored in other slots or in other frames.