

# Narrative

a domain specific language prototype for textual worlds

## Abstract:

Mainstream programming languages are poorly equipped to quickly install a virtual world and directly play with its inhabitants, tools and components. The Narrative language mixes ideas from knowledge representation and theorem proving to offer a lower-barrier entry to simulation in textual virtual worlds.

## Keywords:

knowledge representation, conceptual graphs, simulation, virtual worlds, interactive fictions, textual adventures, puzzles.

## 1. Square boxes

Narrative has no predefined notion such as room, north, south, take. Instead everything is minimalist and built from scratch. The most basic component is the square box. Constant square boxes are `[integer=2]`, `[float=3.1415]`, `[string="hello"]`. A variable square box is `[person]` where person is said to be a concept. The variable square box `[person]` means there exists a person and this person may be different than any other `[person]`. If you want to refer to the specific person `alice` then the square box is `[person:alice]` where `alice` is said to be the referent. A referent name must be unique in the whole world.

New concepts are introduced starting from the upper-most concept via **derive... as... .**

**derive** concept **as** person, city, vehicule, sell.  
**derive** person **as** child, woman, man, sailor.

Taken together the **derive...as...** commands create a concept hierarchy. Using this example hierarchy `[woman:alice]` is compatible with `[person]` and `[person:alice]` because any woman is also a person. Note that `[concept:alice]` holds too because person is a concept. On the contrary `[sailor:alice]` fails because not every woman is a sailor.

## 2. Role arrows, trees, commands

The other component than square boxes is the role arrow.

A role arrow connects a left square box to a right square box like [person:alice]-location->[city:lyon]. Square boxes and role arrows together form a tree. Trees together form a forest that is the textual virtual world.

New trees are created using the **insert** command.

### **insert**

```
[woman:alice]-
  -location->[city:lyon],
  -has->[vehicle:car]-quantity->[integer=2],
  -money->[float=5000.0]
[man:bob]-
  -location->[city:marseille],
  -has->[vehicle:car]-quantity->[integer=1],
  -has->[vehicle:bike]-quantity->[integer=1],
  -money->[float=13000.0].
```

New commands are created using **define...rewrite...as...** where every word beginning with an uppercase letter is a pattern variable within the **define** section and a bound variable within the **as** sections. In the **rewrite** sections there are both old bound variables (bound within the **define** section) and new pattern variables.

```
define [sell]-
  -agent->[person:A],
  -participant->[person:B],
  -theme->T,
  -price->[float=P].
rewrite [person:A]-
  -has->T-quantity->[integer=N],
  -money->[float=M]
as [person:A]-
  -has->T-quantity->[integer=N-1],
  -money->[float=M+P]
rewrite [person:B]-
  -has->T-quantity->[integer=N],
  -money->[float=M]
as [person:B]-
  -has->T-quantity->[integer=N+1],
  -money->[float=M-P].

define [travel]-
  -agent->[person:P],
  -instrument->[vehicle:V],
  -start->[city:A],
  -destination->[city:B]
```

```
rewrite [person:P]-location->[city:A]  
as [person:P]-location->[city:B].
```

Defined commands can be invoked using the **command** keyword.

#### **command**

```
[travel]-  
  -agent->[woman:alice],  
  -instrument->[vehicle:car],  
  -start->[city:lyon],  
  -destination->[city:marseille]  
[sell]-  
  -agent->[woman:alice],  
  -participant->[man:bob],  
  -theme->[vehicle:car],  
  -price->[float=12000.]  
[sell]-  
  -agent->[man:bob],  
  -participant->[woman:alice],  
  -theme->[vehicle:bike],  
  -price->[float=4000.]  
[travel]-  
  -agent->[woman:alice],  
  -instrument->[vehicle:bike],  
  -start->[city:marseille],  
  -destination->[city:lyon].
```

### **3. Deep trees**

The semicolon allows to deepen trees at arbitrary levels.

#### **insert**

```
[woman:alice]-  
  -son->[child:charlie]->  
    -location->[city:lyon],  
    -friend->[child:dylan]->  
      -location->[city:lyon],  
      -has->[vehicle:roller_skate]-quantity->[integer=1],  
      -money->[float=100.0];  
    -has->[vehicle:bicycle]-quantity->[integer=1],  
    -money->[float=300.0];  
  -location->[city:lyon],  
  -has->[vehicle:car]-quantity->[integer=2],  
  -money->[float=5000.0].
```

### **4. Nested trees**

Another (and last) kind of square box is proposition. A proposition is a square

box [concept=...] where ... can be a full-featured tree. Propositions allow complex rules such as Tom believes that Eva wants to marry a sailor.

derive concept as proposition, situation.

```
insert
  [man:tom]-believe->
    [proposition =
      [woman:eva]-want->[situation =
        [woman:eva]-marry->[sailor]
      ]
    ].
```

## 5. More commands and patterns

The **show** command displays the whole world or a chosen tree or sub-tree.

The **select** command displays all sub-trees satisfying the given pattern.

```
select [woman]-son->[child]-R->T.
```

Here R is a role pattern variable and T is a tree pattern variable (it will capture the whole tree at right of R).

## 6. Inversed role arrows

Any role arrow can be right-to-left instead of left-to-right.

Said otherwise you have 2 different ways to tell that alice and charlie are mother and son.

```
insert
  [woman:alice]-son->[child:charlie]
  [woman:alice]<-mother-[child:charlie].
```

```
insert
  [child:charlie]-mother->[woman:alice]
  [child:charlie]<-son-[woman:alice].
```

## 7. Interactive fiction specifics

I am not deep into this stuff at the moment because i want to keep Narractive as general-purpose as possible.

**insert**, **define** and **rewrite** may be too powerful to be allowed by an

interactive fiction player, so you can turn them **off**.

`insert define rewrite off.`

Of course if your main character is (or becomes) an all-mighty wizard or a super-quantum-conscious-computer you can eventually turn them **on**:

`insert define rewrite on.`

How the interactive fiction back-story is told is not yet written in stone. Chances are that a **display** command and a **prompt** command will do the basic job.

```
display "This is the fantasy back-story."  
prompt "WIZARD> ".
```

## 8. Related work

[Dialog](#) is a domain specific language for Interactive Fiction authoring. Dialog has a bunch of features to tightly control the text display. Dialog offers a Prolog-like rule matching engine but is not equipped to handle natural-look-alike language. Despite this limitation Dialog is good enough for typical IF authoring because almost no author claim for advanced text processing.

[Prolog+CG](#) features a Prolog plus Conceptual Graphs. Thus one would imagine Prolog+CG is the ideal Interactive Fiction creator. Unfortunately Prolog+CG lacks support. Trying to [contact](#) the maintainer via email or forum is a waste of time.