A Baginner's Notes on Genesis

Yang Zhutian (Yang)

February 13, 2018

Contents

1 The Basics: Four Classes in a Role Frame

1.1 Entity

1.1.1 Create an Entity

```
// Every instance has a unique name and bundle of threads.
Entity x = new Entity ("ball");
Mark.say("Entity:", x);
Mark.say("Type:", x.getType()); // the default type is "thing" if new Entity ();
Mark.say("Verbose:", x.toXML()); // i.e. entity = name + types
----------------------------- Output
>>> Entity: (ent ball-0)
>>> Type: ball
>>> Verbose:
<entity>
 <name>ball-0</name>
 <bundle><thread>thing ball</thread></bundle>
</entity>
```

1.1.2 Add Type to an Entity

```
Entity y = new Entity ("ball");
y.addType("baseball");
y.addType("riceball");
y.addType("goodevening");
Mark.say("Entity:", y);
Mark.say("Type:", y.getType());
Mark.say("Verbose:", y.toXML());
Mark.say(y.isA("ball"),"It is a ball");
----------------------------- Output
>>> Entity: (ent goodevening-1)
>>> Type: goodevening
>>> Verbose:
<entity>
 <name>goodevening-1</name> // name is the finest level of categorization
 <bundle><thread>thing ball baseball riceball goodevening</thread></bundle>
</entity>
>>> It is a ball
```

1.2 Function

```
// Functions represent Jackendoff's places and path elements
Function f = new Function("in", x);
Mark.say("Entity:", f);
Mark.say("Type:", f.getType());
Mark.say("Verbose:", f.toXML());
---------------------------- Output
>>> Function: (fun in (ent ball-0))
>>> Type: in
>>> Verbose:
<entity>
 <name>in-3</name>
 <bundle><thread>thing in</thread></bundle>
 <subject>
  <entity>
   <name>ball-0</name>
   <bundle><thread>thing ball</thread></bundle>
  </entity>
 </subject>
</entity>
```

1.3 Sequence

1.3.1 Create a Sequence Without Functions

```
Entity Peter = new Entity("Peter");
Entity knife = new Entity("knife");

// Sequence are merged sentence elements
Sequence roles = new Sequence("roles");
roles.addElement(Peter);
roles.addElement(knife);
Mark.say("Sequence:", roles);
Mark.say("Type:", roles.getType());
Mark.say("Verbose:", roles.toXML());

---------------------------- Output
>>> Sequence: (seq roles (ent Peter-8) (ent knife-9)) // these two sequence elementss do not
    make sense
>>> Type: roles
>>> Verbose:
<entity>
 <name>roles-7</name>
 <bundle><thread>thing roles</thread></bundle>
 <sequence>
  <element>
   <entity>
    <name>Peter-8</name>
    <bundle><thread>thing Peter</thread></bundle>
   </entity>
  </element>
  <element>
   <entity>
    <name>knife-9</name>
    <bundle><thread>thing knife</thread></bundle>
   </entity>
  </element>
 </sequence>
</entity>
```

1.4 Relation

1.4.1 Create a Relation with Keywords

```
// Relations adds an object slot to the Function class
// most sentences are relations
Entity h = new Entity("heart");
Entity s = new Entity("skin");
Relation r = new Relation("between",h,s);
Mark.say("Relation:", r);
Mark.say("Type:", r.getType());
Mark.say("Verbose", r.toXML());
---------------------------- Output
// the first is the subject, the second is the object
>>> Relation: (rel between (ent heart-4) (ent skin-5))
>>> Type: between
>>> Verbose
<entity>
 <name>between-6</name>
 <bundle><thread>thing between</thread></bundle>
 <subject>
  <entity>
   <name>heart-4</name>
   <bundle><thread>thing heart</thread></bundle>
  </entity>
 </subject>
 <object>
  <entity>
   <name>skin-5</name>
   <bundle><thread>thing skin</thread></bundle>
  </entity>
 </object>
</entity>
```

1.4.2 Create a Relation With Roles and Functions

```
// first define a way of killing people
Sequence roles2 = new Sequence("roles");
roles2.addElement(new Function("object", new Entity("Peter")));
roles2.addElement(new Function("with", new Entity("knife")));

// then, let John kill it
Relation k2 = new Relation("kill", new Entity("John"), roles2);
Mark.say("Relation:", k2);
---------------------------- Output
>>> (rel kill (ent John-13) (seq roles (fun object (ent Peter-8)) (fun with (ent knife-9))))
```

2 The Structure: Role Frame

2.1 MakeRoleFrame and AddRole

```
Entity rf = RoleFrames.makeRoleFrame(new Entity("John"), "kill", new Entity("Peter"));
Mark.say("Before AddRole:", rf);
rf = RoleFrames.addRole(rf, "with", new Entity("knife"));
Mark.say("After AddRole:", rf);
---------------------------- Output
>>> Before AddRole: (rel kill (ent John-17) (seq roles (fun object (ent Peter-8))))
>>> After AddRole: (rel kill (ent John-17) (seq roles (fun object (ent Peter-8)) (fun with
    (ent knife-9))))
```

2.2 Internalize and Generalize

```
Translator t = Translator.getTranslator();
Generator g = Generator.getGenerator();

// determine if a noun is in the WordNet
Bundle bundle = BundleGenerator.getBundle("Dog");
Mark.say(!bundle.isEmpty(), "Dog in WordNet", bundle);

bundle = BundleGenerator.getBundle("Bouvier"); // not in WorldNet
Mark.say(!bundle.isEmpty(), "Bouvier in WordNet", bundle); // false

// -----------------------------------
// 1st example of specification
t.internalize("Sonja is a bouvier.");
Entity property = t.internalize("Sonja is brave.");
Entity sonja = property.getSubject();
Mark.say("\nBefore Internalize:", sonja.getPrimedThread());

// 2nd example of even more specification
t.internalize("A bouvier is a kind of dog.");
t.internalize("Sonja is a bouvier.");
property = t.internalize("Sonja is brave.");
sonja = property.getSubject();
Mark.say("\nAfter Internalize:", sonja.getPrimedThread());

// -----------------------------------
// 1st time of generalization
Entity generalization = sonja.geneneralize(sonja);
Mark.say("\n1st generalization", generalization.getPrimedThread());

// 2nd time of generalization
generalization = generalization.geneneralize(sonja);
Mark.say("\n2nd generalization", generalization.getPrimedThread());
----------------------------- Output
>>> Dog in WordNet
<bundle>
   <thread>thing entity physical-entity object whole living-thing organism animal chordate
       vertebrate mammal placental carnivore canine dog Dog</thread>
   <thread>thing entity physical-entity object whole living-thing organism person
       unwelcome-person unpleasant-person unpleasant-woman frump dog Dog</thread>
   ...
</bundle>
>>>
Before Internalize:
<thread>thing bouvier name sonja</thread>
>>>
After Internalize:
<thread>thing entity physical-entity object whole living-thing organism animal chordate
    vertebrate mammal placental carnivore canine dog bouvier name sonja</thread>
>>>
1st generalization
<thread>thing entity physical-entity object whole living-thing organism animal chordate
    vertebrate mammal placental carnivore canine dog bouvier</thread>
>>>
2nd generalization
<thread>thing entity physical-entity object whole living-thing organism animal chordate
    vertebrate mammal placental carnivore canine dog</thread>
```

2.3 Substitutor

```
Translator t = Translator.getTranslator();

t.translateToEntity("John is a person"); // the same as internalize???
t.translateToEntity("Mary is a person");
t.translateToEntity("Susan is a person");

Entity john = Start.makeThing("john"); // if comment the above lines, the index will start
    from 0
Entity mary = Start.makeThing("mary"); // if not, starts from 200
Entity susan = Start.makeThing("susan");

Entity roleFrame = RoleFrames.makeRoleFrame(john, "loves", mary);
Mark.say("Original role frame: ", roleFrame);

Entity newRoleFrame = Substitutor.substitute(susan, mary, roleFrame);
Mark.say("Result of substitution", newRoleFrame);

Mark.say("In English:\n",
   Generator.getGenerator().generate(roleFrame), "\n",
   Generator.getGenerator().generate(newRoleFrame), "\n");
---------------------------- Output
>>> Original role frame: (rel loves (ent john-201) (seq roles (fun object (ent mary-202))))
>>> Result of substitution (rel loves (ent john-201) (seq roles (fun object (ent
    susan-203))))
>>> In English:
 John loves Mary.
 John loves Susan.
```

2.4 CreateTrajectory and AddPathElement

```
// entity -> place -> path element -> path -> trajectory
// path elements: "from origin" and "to destination"
// path: a sequence of path elements
// trajectory: "sb. do sth." + path
Entity tree = new Entity("tree");
Function place = JFactory.createPlace("top", tree);
Function origin = JFactory.createPathElement("from", place);

Sequence path = JFactory.createPath();
path.addElement(origin); // Add the origin (path element)
Entity trajectory = JFactory.createTrajectory(Peter, "fly", path);

Function destination = JFactory.createPathElement("to", JFactory.createPlace("at", new
    Entity("rock")));
JFactory.addPathElement(path, destination);

Mark.say("Amended trajectory role frame: " + trajectory);
---------------------------- Output
>>> Amended trajectory role frame: (rel fly (ent Peter-8) (seq roles (fun object (seq path
    (fun from (fun top (ent tree-41))) (fun to (fun at (ent rock-68)))))))
```

3 The Translation: Innerese

3.1 Translator

```
Translator translator = Translator.getTranslator();
Generator generator = Generator.getGenerator();
String sentence = "John marries Mary because John loves money";

// from English to Innerese (role frames)
Entity entity = translator.translate(sentence);
Mark.say("Innerese:", entity);
Mark.say("English:", generator.generate(entity.getElements().get(0)));
---------------------------- Output
>>> Innerese: (seq semantic-interpretation (rel cause (seq conjuction (rel love (ent
    john-106) (seq roles (fun object (ent money-116))))) (rel marry (ent john-106) (seq roles
    (fun object (ent mary-113))))))
>>> English: John marries Mary because John loves money.
```

3.2 Generator

```
// Make a role frame for demonstration
Entity entity = RoleFrames.makeRoleFrame("John", "love", "Mary");
String english = Generator.getGenerator().generate(entity);
Mark.say(english);

// More complicated
Entity entity2 = RoleFrames.makeRoleFrame("John", "want", entity);
english = Generator.getGenerator().generate(entity2);
Mark.say(english);
---------------------------- Output
>>> John loves Mary.
>>> John wants to love Mary.
```

3.3 Translate to Entity

```
Translator tt = Translator.getTranslator();
Entity entity = tt.translateToEntity("John loves Mary");
s = entity.getSubject();
Mark.say("Before Imparting:", s.toXML());

tt.translateToEntity("John is a person");
ss = ee.getSubject();
Mark.say("After Imparting:", ss.toXML());

tt.translateToEntity("John is a person");
ee = tt.translateToEntity("John loves Mary"); // the same as tt.internalize("John ...");
ss = ee.getSubject();
Mark.say("The order matters:", ss.toXML());

tt.translateToEntity("John is a person");
ee = tt.internalize("John loves Mary");
ss = ee.getSubject();
Mark.say("The order matters:", ss.toXML());

---------------------------- Output
>>> Before Imparting:
<entity>
 <name>john-90</name>
 <bundle><thread>thing entity physical-entity object whole artifact structure area room
     toilet john name john</thread> // now John might be a toilet
  <thread>thing John john</thread>
  <thread>thing entity physical-entity object whole living-thing organism person user
      consumer customer whoremaster john</thread></bundle>
</entity>
>>> After Imparting:
<entity>
 <name>john-90</name> // notice it is the same john
 <bundle><thread>thing entity physical-entity object whole artifact structure area room
     toilet john name john</thread>
  <thread>thing John john</thread>
  <thread>thing entity physical-entity object whole living-thing organism person user
      consumer customer whoremaster john</thread></bundle>
</entity>
>>> The order matters:
<entity>
 <name>john-214</name> // notice it is a different john
 <bundle><thread>thing entity physical-entity object whole living-thing organism person name
     john</thread></bundle>
</entity>
```

4 The Understanding

4.1 Roles

4.1.1 Object by x.getObject()

```
x = t.translateToEntity("George is husband of Mary"); // Mary is the subject here
Mark.say("Entity:", x);
Mark.say("Entity type:", x.getType());
Entity s = x.getSubject();
Entity o = x.getObject();
Mark.say("Subject and object:", s, o);
----------------------------- Output
>>> Entity: (rel husband (ent mary-150) (ent george-137))
>>> Entity type: husband
>>> Subject and object: (ent mary-150) (ent george-137)
```

4.1.2 Object by RoleFrames.getObject(x)

```
x = t.translateToEntity("George is killed by Mary"); // Mary is the subject here
Mark.say("Entity:", x);
s = x.getSubject();
o = x.getObject();
Mark.say("Subject and object:", s, o);
o = RoleFrames.getObject(x);           //  It is a role frame. Use special getters:
Mark.say("Subject and object:", s, o);
----------------------------- Output
>>> Entity: (rel kill (ent mary-200) (seq roles (fun object (ent george-219))))
>>> Subject and object: (ent mary-200) (seq roles (fun object (ent george-219)))
>>> Subject and object: (ent mary-200) (ent george-219)
```

4.1.3 Owner by s.getProperty(Markers...)

```
Entity x = t.translateToEntity("Paul's army fled.");
Mark.say("Entity:", x);
Mark.say("Entity type:", x.getType());

Entity subject = x.getSubject();
Mark.say("Subject:", subject);
Mark.say("Owner:", subject.getProperty(Markers.OWNER_MARKER));
----------------------------- Output
>>> Entity: (rel flee (ent army-90) (seq roles))
>>> Entity type: flee
>>> Subject: (ent army-90)
>>> Owner: (ent paul-98)
```

4.1.4 Tools by RoleFrames.getRoles("with", x)

```
x = t.translateToEntity("George helped Mary with Peter with insights");
List<Entity> roles = RoleFrames.getRoles("with", x);
roles.stream().forEachOrdered(role -> Mark.say("Role with \"with\" preposition:", role));
---------------------------- Output
>>> Role with "with" preposition: (ent peter-284)
>>> Role with "with" preposition: (ent insights-288)
```

4.2 Types

4.2.1 Check Type of Entity

```
// translator will be discussed later
Translator t = Translator.getTranslator();
x = t.translateToEntity("George helped Mary with Peter with insights");

// (1) getType (2) getPrimedThread get the entire primed thread
Mark.say("Type:", x.getType());
Mark.say("Primed thread:", x.getPrimedThread());
// Print everything in the thread:
x.getPrimedThread().stream().forEachOrdered(c -> Mark.say("It is a:", c));
---------------------------- Output
>>> Type: help
>>> Primed thread:
<thread>action support help</thread>
>>> It is a: action
>>> It is a: support
>>> It is a: help
```

4.2.2 Check Type of Sentence/Rolation

```
Translator t = Translator.getTranslator();

x = t.translateToEntity("A bouvier is a kind of dog");
Mark.say("Entity:", x);
Mark.say("Entity type:", x.getType());

x = t.translateToEntity("John is fat");
Mark.say("Entity:", x);
Mark.say("Entity type:", x.getType());
---------------------------- Output
>>> Entity: (rel classification (ent dog-347) (ent bouvier-339))
>>> Entity type: classification
>>> Entity: (rel property (ent john-385) (seq roles (fun object (ent fat-405))))
>>> Entity type: property
```

5 The Inference

5.1 By Deduction Rules

```
Translator t = Translator.getTranslator();
// Get everything to have appropriate class
t.internalize("xx is a person.");
t.internalize("yy is a person.");
t.internalize("Peter is a person");
t.internalize("Paul is a person.");

Entity rule = t.internalize("If xx kills yy, then yy is dead");
Entity antecedent = rule.getSubject().getElements().get(0);
Entity consequent = rule.getObject();
Mark.say("Rule", rule);
Mark.say("Antecedent", antecedent);
Mark.say("Consequent", consequent);

Entity event = t.internalize("Peter kills Paul");
Mark.say("Event", event);

// Match antecedent with event, produce binding list
LList<PairOfEntities> bindings = StandardMatcher.getBasicMatcher().match(antecedent, event);
Mark.say("Bindings", bindings);

// Now, substitute into consequent using bindings
Entity result = Substitutor.substitute(consequent, bindings);
Mark.say("Result", result);
---------------------------- Output
>>> Rule (rel cause (seq conjuction (rel kill (ent xx-286) (seq roles (fun object (ent
    yy-260))))) (rel property (ent yy-260) (seq roles (fun object (ent dead-262)))))
>>> Antecedent (rel kill (ent xx-286) (seq roles (fun object (ent yy-260))))
>>> Consequent (rel property (ent yy-260) (seq roles (fun object (ent dead-262))))
>>> Event (rel kill (ent peter-358) (seq roles (fun object (ent paul-360))))
>>> Bindings (<xx-286, peter-358> <yy-260, paul-360>)
>>> Result (rel property (ent paul-360) (seq roles (fun object (ent dead-262))))
```

5.2 By Mental Model

```
// import models.MentalModel;
MentalModel mm = new MentalModel("Amanda", "eastern.txt", true);
Mark.say("The number of rules in the model is",
    mm.getStoryProcessor().getRuleMemory().getRuleList().size());
Mark.say("The number of concepts in the model is",
    mm.getStoryProcessor().getConcepts().size());

---------------------------- Output
>>> The number of rules in the model is 4
>>> The number of concepts in the model is 1
```

6 The Pattern

6.1 Compare Two Sentences

```
Translator t = Translator.getTranslator();
t.translate("John is a person");
t.translate("Mary is a person");
t.translate("Susan is a person");
t.translate("Peter is a person");
t.translate("Paul is a person");
// if I delete ant of these lines, both similarity and difference will give "null"

Entity X = t.translateToEntity("John stabbed Mary with Susan.");
Entity Y = t.translateToEntity("Peter stabbed Patrick with Paul with a knife at night.");

LList<PairOfEntities> matches = StandardMatcher.getBasicMatcher().match(X, Y);
Mark.say("Similarity between X and Y", matches);

List<Entity> excess = StandardMatcher.getBasicMatcher().matchAndReportExcess(X, Y
Mark.say("Difference between X and Y", excess);
---------------------------- Output
>>> Similarity between X and Y (<john-279, peter-334> <susan-290, paul-339> <mary-287,
    patrick-336>) : zhutianYang.School.TestMatchInnerese.main(TestMatchInnerese.java:33)
>>> Difference between X and Y [(fun with (ent knife-341)), (fun at (ent night-349))] :
    zhutianYang.School.TestMatchInnerese.main(TestMatchInnerese.java:36)
```