Learning to Read Recipes : Activity Diagramming with Narrative Event Chains

A Thesis Presented

by

Ganesa Thandavam Ponnuraj

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

December 2014

Stony Brook University

The Graduate School

Ganesa Thandavam Ponnuraj

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis

Prof. Yejin Choi – Thesis Advisor
Assistant Professor, Department of Computer Science and Engineering, University of Washington

Prof. Steven Skiena – Chairperson of Defense
Distinguished Teaching Professor, Department of Computer Science, Stony Brook University

Prof. Paul Fodor
Research Assistant Professor, Department of Computer Science, Stony Brook University

This thesis is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

Learning to Read Recipes : Activity Diagramming with Narrative Event Chains

by

Ganesa Thandavam Ponnuraj

Master of Science

in

Computer Science

Stony Brook University

2014

In this work, we generate activity diagrams of the recipe text automatically, with the help of narrative event chains. Cooking recipes, that have predominantly imperative sentences, present unique challenges in the form of frequent argument drops and co-reference resolution over evolving (or merging) entities. Here, we introduce an unsupervised approach to automatically recover the *recipe flow graphs*. Then we illustrate the usefulness of our learned models via narrative *cloze* task; for the automatic evaluation of our learned models, we make use of one of the oft-used task in Discourse analysis, namely the *sentence re-ordering* and illustrate their utility. To report the effectiveness of our diagramming algorithm, we report the Precision-Recall metric based on the small set of gold-standard annotations that we made.

To evaluate a diagramming algorithm on a large scale we need a lot of human annotations. To this end we propose a design for a Game with a purpose (GWAP) that could help us in crowd-sourcing the human annotations. We describe the game design with regards to

ease of game play and the considerations that could be taken to keep the player motivated. This game could be used to build a richly annotated recipes corpus. We leave the task of building the game as a future work.

Dedicated to my parents and my siblings.

Contents

List of Figures

List of Tables

Acknowledgements

The 2 rewarding years of my life at the Stony Brook University has been a great learning experience that taught me not just the virtue of patience and persistence but also helped me understand myself better. I am grateful for getting this opportunity to meet and interact with people with great talents from different walks of life, different parts of the world. My sincere thanks to Professor Yejin Choi who guided me, patiently, through the ups and downs during the research project. I feel lucky that I got a chance to work with Yejin who always encouraged me and gave me enough space to learn and grow. The lessons learnt during the research, not just the technical lessons - life lessons too, truly will have a positive bearing in my life going forward, I am sure.

My heartfelt thanks to Polina Kuznetsova who played the role of a mentor while collaborating on my thesis project. I also thank members of the Stony Brook NLP reading group Song Feng, Ritwik Banerjee, Jun Seok Kang, Jianfu Chen and Vikas Ashok for the numerous paper reading sessions and discussions.

Many thanks to Chloe Kiddon and Professor Luke Zettlemoyer for useful comments and discussions.

I also extend my thanks to my mentor Mukarram, my brother Jothi, my sister Shree Priya and my parents Ponnuraj appa and Rani amma for backing me throughout as I embarked on this memorable graduate school life.

No graduate school life without friends. Finally, I would like to thank all my friends who were a constant source of support and inspiration.

Chapter 1

Introduction

1.1 Motivation

Semantic parsing is the process of mapping a natural-language sentence into a formal representation of its meaning. In this work we recover one such representation for recipe graphs namely the *flow graph*. Through this representation, we can observe the partial order relations existing between the different event chains in any recipe, which naturally gives us a graph plan for the cooking tasks.

Imagine the problems one would come across when trying to build a chef-robot that could carry out recipe instructions and make any dish. If we think about a robot carrying out text instructions given to it, we give it the context of semantic parsing required to help the robot understand text instructions. It is this problem, which forefronts task of building a *chef-robot* that can understand recipes in natural language, we attack in this thesis. We also provide the corpus that we curated from web for our experiments, to the community.

Now let us take a look at the recipe instructions for *Banana Crumb Muffins* [1].

1. ....
2. In a large bowl, mix together 1 1/2 cups flour, baking soda, baking powder and salt. In another bowl, beat together bananas, sugar, egg and melted butter. Stir the **banana mixture** into the **flour**

[1] Recipe text parsed from `http://allrecipes.com/Recipe/Banana-Crumb-Muffins/`

mixture just until moistened. Spoon **batter** into prepared muffin cups.

3. ....

Figure 1.1 is the graph generated by our approach for *Banana Crumb Muffins* recipe. One interesting point to talk about in this graph is the fact that we were able to identify an interesting relation that is not evident in the recipe text directly:

- The fact that "stirring" *banana mixture* and *flour mixture* gives the *batter*. We will refer to this phenomenon as "Evolving" entities. In other words, *banana mixture* and *flour mixture* have evolved to become the "batter", upon "stirring".

Similarly, if we look at Figure 1.2 and recipe instructions for *Baked-mac-and-cheese-for-one* recipe [2], we will identify the following:

1. Preheat an oven to 400 degrees F (200 degrees C). Grease an oven-proof soup crock or 1 cup baking dish.
2. ....
3. ....
4. Bake, uncovered, until the cheese is melted and the macaroni is heated through, about 10 minutes.

- The fact that *"bake"* action depends on "preheated oven". We will refer this phenomenon as *Argument Drop*

We will see more about these linguistic phenomena in Chapter 3.

[2]Recipe text parsed from `http://allrecipes.com/Recipe/Baked-Mac-and-Cheese-for-One/`

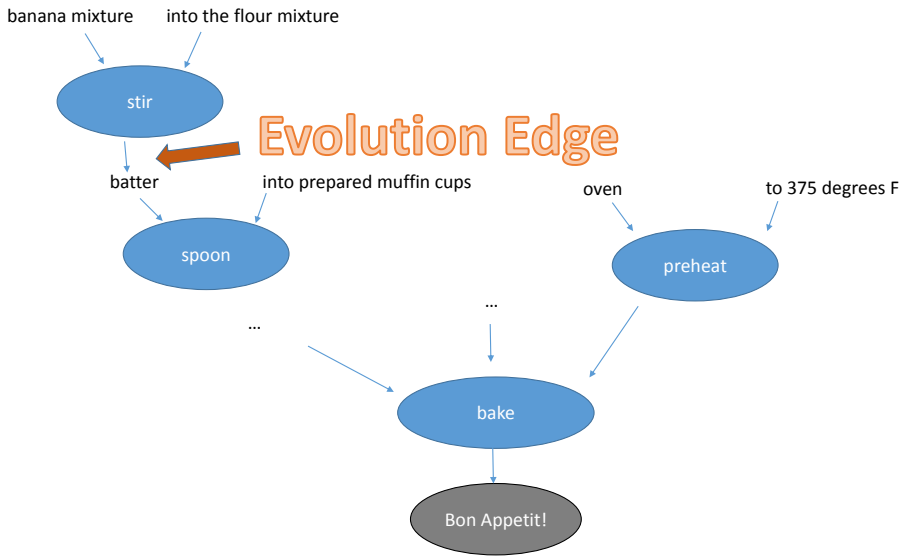

Figure 1.1: Recipe flow graph generated for the Banana-Crumb-Muffins recipe

1.2 Thesis Outline

We will review literature in Chapter 2, talk about our graph representation and linguistic phenomena in Chapter 3, describe out data-set and method in Chapter 4, mention about our experimental setup and results in Chapter 5 and finally give our conclusions and direction for future work in Chapter 6.

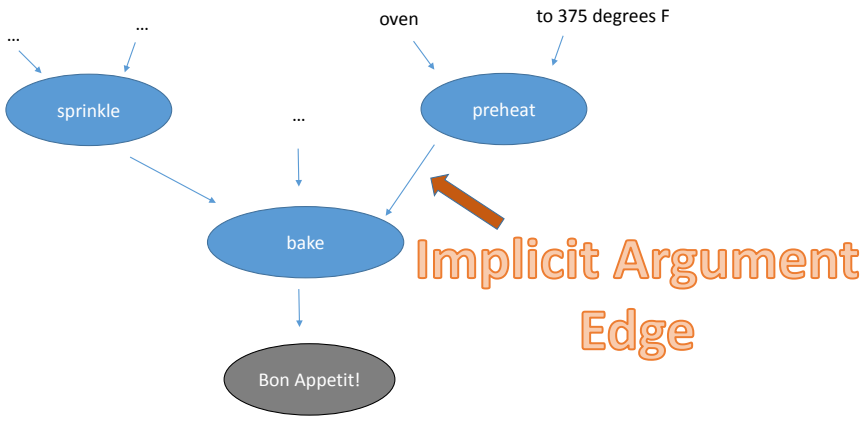

Figure 1.2: Recipe flow graph generated for the Baked-Mac-and-Cheese-for-One recipe

Chapter 2

Related Work

The seminal work of [1] introduced the idea of capturing partial order relations among events which they called as the narrative chains. However, in cooking recipes, because we have to deal with argument drops, learning such chains become challenging. [2] did a thorough study of *implicit arguments* (otherwise called as *null instantiations* by [3]) for nominal predicates. The idea central to [2] was about identifying coreference chains of a candidate implicit argument (otherwise also referred by [2] as *extra-sentential arguments*).

However [2] provide a supervised learning scheme to identify implicit arguments and thereby improve argument coverage in Nombank; whereas we propose an unsupervised approach for recovering these implicit arguments. Additionally, the recipes corpus presents us a unique challenge of dealing with evolution of participating entities through the procedural narrative chains. for eg: a typical procedural narrative chain for a Mac-and-cheese recipe looks like

- add water to a pan
- boil water [*in the pan* is implicit]
- add pasta [*added to boiling water* is implicit]
- cook pasta
- drain [*pasta* is implicit]
- add cheese sauce over cooked pasta [*pasta* over the course of time has evolved to become *cooked pasta*].

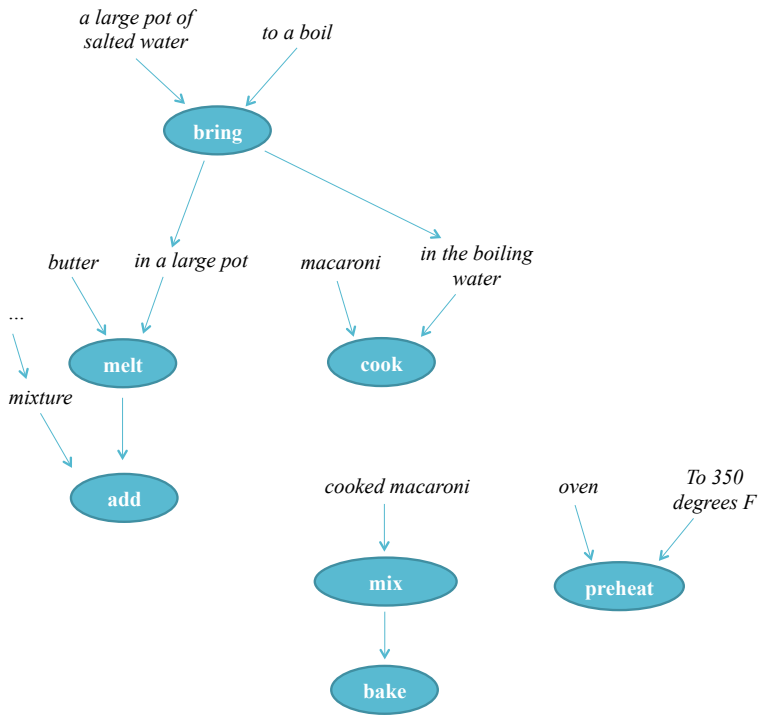

Figure 2.1: Sample output from the coreference resolver

In both [1] and [2], the coreference resolver plays a crucial role. But, the state of the art coreference resolution system by [4] fails to capture the evolution of entities in the recipes corpus, because it is against the *word inclusion* principle, based on which 3 of their 7 sieves were built. According to [5] "word inclusion principle exploits the property of discourse that it is uncommon to introduce novel information in later mentions.".

More appropriate way of dealing with evolution of entities would be to frame the problem as joint entity and event co-reference resolution. Eventhough [6] came up with one such approach for entity and event coreference resolution, their system also relies on the *word inclusion* principle; also their system is not publicly available. Moreover for the purposes of recipe corpus, we do not need elaborate features like gender, animacy, named entity recognition etc.., to resolve co-reference. Hence we apply simplified set of rules, some of which we borrow from [4] and [6], to help us identify procedural narrative chains.

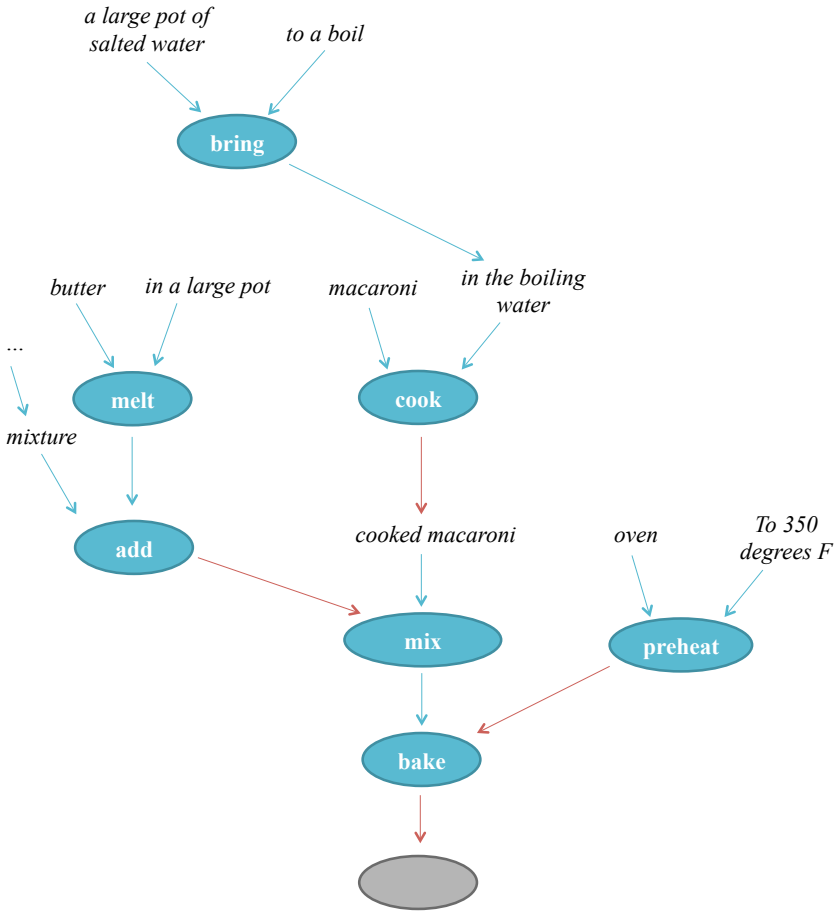

Figure 2.2: The resulting arborescence

[7] study the problem of identifying scripts with multiple arguments, similar to our work. [7] also give a summary of the problems one would face in building content-models for multiple-argument events. However, we differ from [7] in the following aspects:

1. We study the multiple argument events in the cooking recipes dataset that has imperative event chains with linguistic phenomena that is unique to procedural narratives as detailed in Chapter 3

2. We build content-models that make use of 3-skip bigram modelling (similar in spirit to [8]) to recover the activity diagram for the recipe by considering only nouns that occur in *arg1* or *arg2* as explained in Chapter 4. This is a reasonable approximation as we will see in Chapter 4.

Implicit Argument Heuristics

Furthermore, we deal with the implicit argument identification task in the recipes corpus, in 2 ways:

1. We use a heuristic that the implicit argument is always the result of the immediately preceding predicate, with the precedence relation between the predicates determined by the textual order. This assumption works for most cases in the recipes corpus.

2. Secondly, we build models from related recipes to aid implicit argument resolution. By formulating the diagramming problem as that of finding a minimum arborescence, with edges having weights given by these models, we correct the errors made by the coreference resolver and also identify the implicit arguments that are not identified by our first heuristic. For example, refer (Figure 2.1) and (Figure 2.2) to observe the following:

   (a) deletion of an edge from predicate "bring" to argument "in a large pot" (correction of error made by co-reference resolver)

   (b) "preheated oven" is identified as an (implicit) argument for the verb "bake" (arg-drop resolution)

Chapter 3

Recipe Flow Graph and Linguistic Phenomena in Recipes

3.1 Recipe Flow Graph

A finite collection of objects which are related with a partial ordering are said to form a *directed acyclic graph*, by definition. [1] show that narrative chains generally have a partial order relation among them. Therefore we capture this information in the form of a directed acyclic graph. This representation is similar in principle to the representation used by [9]. [9] have used the representation to annotate a corpus whereas we propose an unsupervised approach to recover such a graph from the recipe text automatically. Like [9], we make use of a sentinel node at the bottom that connects different event chains in the recipe text and unifies them. For generating visualizations we use graphviz ([10]).

3.2 Linguistic Phenomena

We assume the following predicate-argument template for capturing cooking knowledge from the corpus. The terminologies are taken from Propbank ([11])

1. Each predicate is assumed to have at-most 2 arguments
2. arg_1 (optional)
3. arg_2 (optional)

Implicit Arguments

Now, let us take a look at a sequence of actions mentioned in a Mac and Cheese recipe namely "*tasty-baked-mac-and-cheese*"[1].

1. Bring a large pot of water to a boil.
2. Cook elbow macaroni in the boiling water, stirring occasionally until almost cooked through and firm to the bite, about 7 minutes.
3. Drain and transfer to a large bowl.
4. ...

If we look at the 3rd step in isolation, we will notice the argument drop for drain (both arg_1 and arg_2 are missing) and transfer (only arg_1 is missing) predicates. We can also notice that the arguments lost are usually either arg_1 or arg_2 in the predicates that occurred immediately before them. This is the intuition behind our simplified heuristic for implicit argument identification. It is important to mention that for the predicate "Drain", when we resolve implicit arguments, we resolve for only arg_1. As a side-effect we will have more arguments of type arg_1 than of type arg_2 in our model.

Evolving Entities

Also, for the purpose of building the recipe flow graph, we look at the arguments in a slightly different manner. In the above example we consider arg_1 of *Drain* to be the output of the *stir* predicate. Similarly, implicit arg1 of *stir* predicate is the output of *cook*.

This results in the emergence of the following chain for the elbow macaroni:

elbow macaroni ⇒ cook ⇒ stir ⇒ drain ⇒ transfer

This kind of argument resolution is slightly different from the way co-reference chains are handled in the literature. It is this phenomenon that we call as *evolving* discourse entities.

[1]Recipe text available at `http://allrecipes.com/Recipe/Tasty-Baked-Mac-n-Cheese/`

Chapter 4

Dataset and Method

4.1 Dataset

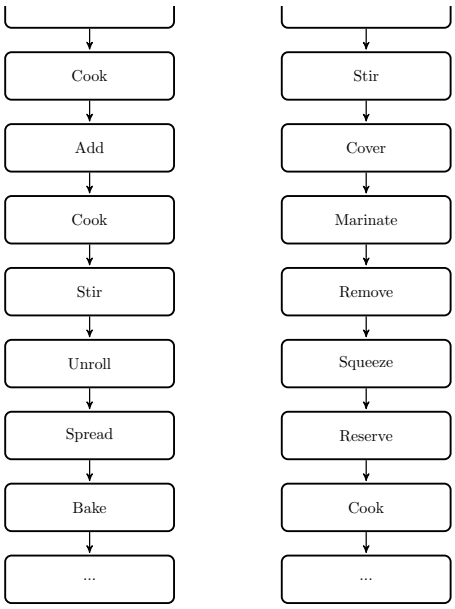

Figure 4.1: Sample Event Chains recovered by our approach Left: A typical event chain that "toppings" participate in the dish "Veggie Pizza" Right: A typical event chain that "chicken" participates in the dish "Chicken Stir Fry"

To capture the linguistic phenomena explained in detail in Chapter 3, we would benefit from having a data-set that truly reflects the event chains that are unique to a particular dish. To this end, we parsed data from `http://allrecipes.com`; the unique aspect of this data-set is the fact that there

are several recipes for preparing a dish. From our learnt models we show the different prominent event chains per dish, refer figure 4.1.

Dish Name	No of Recipes
BananaMuffins	66
BeefChilli	212
BeefMeatLoaf	186
BeefStroganoff	47
CarrotCake	52
CheeseBurger	63
ChickenSalad	302
ChickenStirFry	92
Coleslaw	145
CornChowder	70
DeviledEggs	91
EggNoodles	278
FrenchToast	86
MacAndCheese	100
MeatLasagna	101
PecanPie	81
PotatoSalad	196
PulledPork	70
PumpkinPie	121
VeggiePizza	100

Table 4.1: Recipes Event Chain Dataset

It is because the event chains vary from dish to dish, as shown in figure 4.1, that we want more recipes per dish to build the content-models. The data-set is available here for download All Recipes Data-set,

4.2 Method

We present an unsupervised learning algorithm for recovering these diagrams. The algorithm 1 paints a bird's eye-view of the scheme of things. We will take a look at each step now.

Tregex Expression	Explanation
$VP! >> SBAR[<,VBP \mid <,VB][[< NP\$--PP] \mid [< NP < PP] \mid [< NP] \mid [< PP]\|[<,VBP] \mid [<,VB]]$	Tregex used for parsing; explanation in the following lines.
$VP! >> SBAR$	Verbal phrases that are not a part of sub-ordinate clause
$<,VBP \mid <,VB$	Verbal phrases having "VBP" or "VB" forms of the verb as its head
$< NP\$--PP$	Handling cases when $arg1$ and/or $arg2$ can be absent
$< NP < PP$	
$< NP$	
$< PP$	
$<,VBP$	
$<,VB$	

Table 4.2: Semantic Parsing using Tregex

Algorithm 1 Unsupervised Activity Diagramming for a dish

```
1: for All recipes for the dish do                          ▷ Pre-processing
2:     Identify relevant predicates and their arguments     ▷ semantic parsing
3:     Run co-reference resolver from high precision to low precision
4:     Build content models based on co-reference resolver output
5: end for

6: for iter ← 1, n do                                       ▷ iterative-learning
7:     for All recipes for the dish do
8:         if iter = 1 then
9:             contentModel ← model from line 4
10:        else
11:            contentModel ← model saved from previous iteration
12:        end if
13:        Run arborescence algorithm using contentModel
14:    end for
15:    Build content models based on arborescence output and save for next
   iteration
16: end for
```

4.2.1 Tregex Semantic Parsing

To build a recipe flow graph, as a first step, we need to identify the predicates and their participating arguments, otherwise known as (*semantic parsing*).

We use the Tregex [12] mentioned in Table 4.2 to extract our simplified templates $<$ *verb, NP, PP* $>$, and $<$*PP, verb, NP* $>$ from the stanford PCFG parser [13] output.

We consider all VPs that are not part of a SBAR. We make this assumption because we want to capture main cooking actions without going into details. To capture the relevant imperative cooking actions, we consider only VBP and VB forms of the verbs participating in the recipe text.

4.2.2 Coreference-Resolution Heuristics

In our recipe corpus we have largely imperative narrative chains; also we deal only with utensils, ingredients, intermediate results in a recipe etc.., As such, like already mentioned, features like animacy, gender, named entity recognition etc.., are not useful here. Therefore, in our simplified co-reference resolution model, we rely on the following heuristics, for the purpose of capturing procedural narrative chain:

1. Implicit argument resolution (IArg)
2. Derivationally Related form (Der. Reln)
3. String match of nouns in the participating lexical chains (ArgString match)

Since the entities may evolve with each step in a recipe narrative, to resolve co-referring entities, at each predicate we apply rules on nodes preceding the predicate in the reverse textual order. We stop once we identify a candidate antecedent, since we keep track of the lexical argument chains at each predicate.

Similar in spirit to [4], we run the rules from high precision (IArg) to low precision (ArgString match). By evaluating our diagrams using different ordering of heuristics, having IArg heuristic as the first rule yielded highest precision and also recall. This is in agreement with our observation that most of the times, the implicit arguments are present in the immediately preceding sentence.

Once we apply co-reference resolution heuristics, we get islands of connected components. It is because some of the argument slots are not filled by our coreference resolver that we end up having a forest. For example (Figure 4.2). Therefore to connect the islands in the forest into a single unified tree, we formulate the problem as that of finding min-weight arborescence.

4.2.3 Arborescence as a Natural Recipe Flow

Output from the co-reference resolver is not yet suitable for activity diagramming for 2 reasons.

- Firstly, it contains parallel edges. Figure 4.2 shows an example of such edges. Predicate "bring" can have either "large pot" or "boiling" water as its output *(*artifact of the coreference resolution heuristics). These two possible outcomes represent two completely different branches in the recipe instruction chart. One is related to "cook macaroni", another one to "melt butter". From the original recipe text a person can easily infer that "bring" should be connected to "boiling water".
- Secondly, in Figure 4.2 one can see several connected components of the graph, not connected to each other (incomplete diagram). For example, the component corresponding to "preheating the oven", is isolated from the rest of the graph.

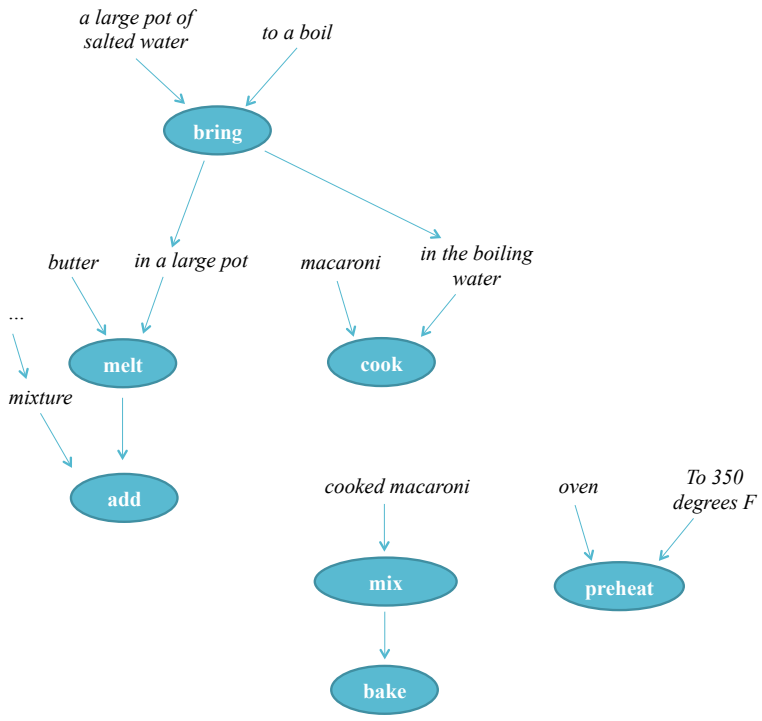

Figure 4.2: Sample output from the coreference resolver

Thus, we need to both, delete ambiguous edges and introduce additional ones. But which nodes must the additional edges connect and which edges should we delete ? Let us answer this in the following section.

4.2.4 Arborescence formulation

Denote flow graph as G (Figure 4.2). We introduce additional edges between connected components of graph G (Figure 4.3). For each connected component we find all nodes with 0 out-degree. We call those nodes *bottom nodes* as they are at the bottom of the connected component output by the coreference resolver. It is worth noting that only predicates play the role of a bottom node. From each *bottom node* we draw an edge leading to *particular* nodes in other connected components. A destination node can be of one of the two categories.

1. The first category includes any predicate

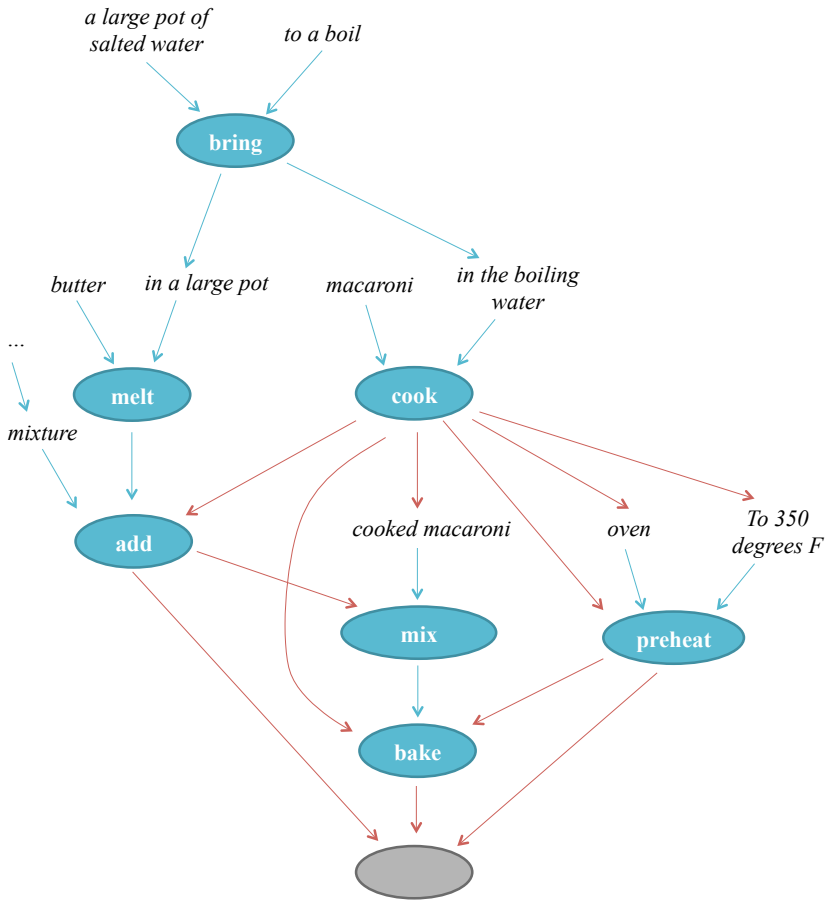

Figure 4.3: Graph (input to arborescence algorithm) with extra edges (in red) and a gray sentinel node

2. The second category consists of only arguments represented by nodes with 0 in-degree. An argument can be of any type i.e arg1 or arg2

We also introduce an additional ''sentinel" node, which represents the final output of the recipe and connect all nodes with 0 out-degree to it. The resulting graph we denote as G'. Then arborescence of the reverse of the graph G' is a reverse flow graph of the recipe.

4.2.5 Content Model for Multiple Argument Events in Recipes

Because of our semantic parsing explained in Section 4.2.1, we will have phrases in *arg1* and *arg2* positions. It is worth noting that in recipe text we will

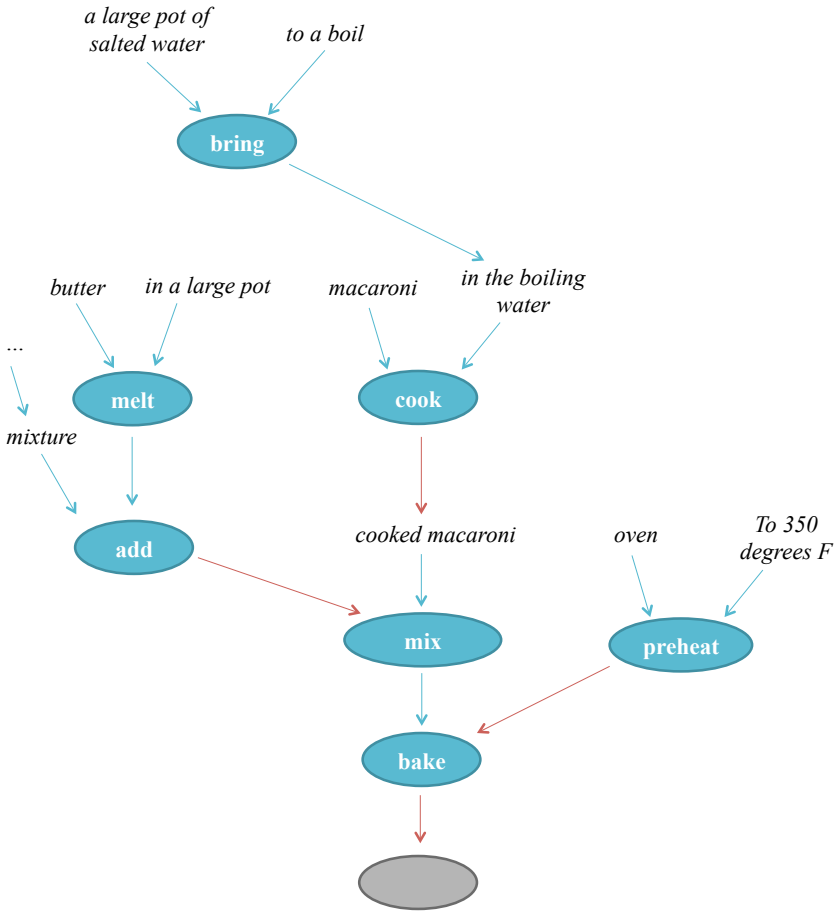

Figure 4.4: The resulting arborescence

have either ingredients or intermediate results (noun phrases) or utensils or duration appearing in these slots. So to model them we consider only the distinct nouns present in these positions. We build models based on counts mentioned in Table 4.3. We use 2-skip bigram based counts similar in spirit to [8], to account for data-sparsity. There are 2 kinds of edges that we will have:

1. predicate to predicate edge: We call them implicit argument edge
2. predicate to (output)argument edge: We call them evolution edge

We assign weights $(1 - P(.))$ to all edges in the graph using our content models (refer Table 4.3) and find minimum weight arborescence.

We apply ChuLiu/Edmonds' algorithm [14] and [15] for finding the resulting arborescence. The algorithm is run on the reversed G' and then the final

Edge Type	Value considered
Evolution	$P(edge(v, output\_arg) \mid arg(1or2), output\_arg)$
	$P(edge(v, output\_arg) \mid v, v\_arg1, v\_arg2, output\_arg)$
	$P(edge(v, output\_arg) \mid v, v\_arg1, output\_arg)$
	$P(edge(v, output\_arg) \mid v\_arg1, v\_arg2, output\_arg)$
	$P(edge(v, output\_arg) \mid v\_arg1, output\_arg)$
Implicit	$P(edge(v1, v2) \mid v1\_arg1, v1, v2, v2\_arg(1or2))$
	$P(edge(v1, v2) \mid v1\_arg1, v1\_arg2, v1, v2, v2\_arg(1or2))$
	$P(edge(v1, v2) \mid v1\_arg1, v1, v2)$
	$P(edge(v1, v2) \mid v1\_arg1, v1\_arg2, v1, v2)$
	$P(edge(v1, v2) \mid v1\_arg(1or2), v1)$
	$P(edge(v1, v2) \mid v1, v2)$

Table 4.3: Content Model for edge weight assignment

result is obtained by taking the reverse graph of arborescence. An example of the result shown at Figure 4.4.

Chapter 5

Experimental Setup and Results

We evaluate the utility of our content models and our unsupervised approach using 3 different experiments. We present precision-recall P/R metric for the diagramming algorithms in section 5.1, elaborate on the sentence re-ordering experiment and present results for the same in section 5.2, and finally present results for the narrative *cloze* task that is used traditionally to evaluate content models that try to capture narrative event chains, in section 5.3.

5.1 Precision Recall for Diagramming

To demonstrate the utility of our diagramming algorithm, we generate diagrams using the following approaches and report the Precision and Recall metric for the edges identified by our algorithms. To report this metric we need some gold standard annotations. For this purpose we manually annotated a small subset of recipes. To get these annotations on a large scale, we propose a game in Chapter 6.

The different approaches for diagramming are :

1. 5 iterations of Algorithm 2
2. 2 iterations of Algorithm 2
3. 5 iterations of Algorithm 2 without co-reference resolution
4. Single Iteration. Using Text order instead of Content Model in Algorithm 2 without co-reference resolution

The annotations that we used for evaluating the algorithms are available here Gold Standard Annotations

Method	Recall - Implicit Arguments	Recall - Evolution	Recall All Edges	Precision All Edges
Coref + 5 Iterations	0.40	0.23	0.51	0.63
Coref + Single Iteration	0.40	0.21	0.51	0.62
No Coref + 5 iterations	0.22	0.14	0.44	0.55
No Coref + Text Order	0.17	0.07	0.42	0.52

Table 5.1: Precision and Recall for Diagramming algorithms

5.2 Viterbi sentence order decoding

The process of identifying the coherent order of sentences among the given sentences is called the sentence re-ordering task. Several experiments have been conducted in the past ([16] and [17]) to show how *Centering* theory [18] could be used to identify sentence ordering. This experiment is also used as a metric for text-to-text generation tasks [19]. We introduce this experiment to automatically evaluate content models that learn narrative event chains. We use our content models and recover the viterbi style sentence order. For comparison we use 2 simple baselines based on our Algorithm 1

5.2.1 Generating Training/Testing Samples

For this task, among all the (predicate, arg1, arg2) groups - henceforth called sem-group, we generate all possible pairs $\binom{N}{2}$. Based on a coin-flip we generate a + or a - negative sample, type of which is determined by the precedence relation in the recipe-text. For example :

1. Sample(sem-group i-1, sem-group i) is a positive pair

2. Sample(sem-group i+1, sem-group i) is a negative pair

Now we train a Linear kernel in SVM [20], using unigram features in sem-group, to predict the precedence relation between given pair of sem-groups.

Algorithm 2 Viterbi Sequence Finder

```
1: procedure getPathProbability(path, transitions, next)
2:     ret ← 0
3:     for i ← 1, length(path) do                    ▷ adding log probabilities
4:         ret ← ret + transitions[path[i]][next]
5:     end for
6:     for i ← 1, length(path) do ▷ accounting for all the following nodes in the path
7:         if i <> next, i ∉ path then
8:             ret ← ret + transitions[next][path[i]]
9:         end if
10:    end for
11:    return ret
12: end procedure

13: procedure getViterbiSequence(transitions, n)    ▷ returns the max probable sequence
14:    for i ← 1, n do                                ▷ First viterbi iteration
15:        path[i] ← [i]                              ▷ Initializing, for starting state
16:        if i = 1 then
17:            Initialize sentence 1 with max probability
18:        else
19:            Initialize other sentences with miniscule probability
20:        end if
21:    end for
22:    for t ← 2, n do                ▷ Remaining viterbi iterations, from 2 to n
23:        for s1 ← 1, n do
24:            for s0 ← 1, n do
25:                candidates ← candidates.add((viterbi[t − 1][s0] + addPathProbability(path[s0], s1, transitions), s1))
26:            end for
27:            (maxProb, maxState) ← max(candidates)
28:            viterbi[t][s1] ← maxProb
29:            newPath[y] ← path[maxState] + s1
30:        end for
31:        path ← newPath
32:    end for
33:    maxState ← argmax(viterbiTable[t][x])
34:    return path[maxState]                          ▷ max probable viterbi sequence
35: end procedure
```

Metric	Binary Classification Precision			Kendall's τ	
Method:	SVM (MAP)	method 1	method 2	method 1	method 2
BananaMuffins	76.86	77.44	82.56	0.50	0.70
BeefChilli	74.68	76.09	72.43	0.62	0.54
BeefMeatLoaf	77.23	78.13	70.99	0.67	0.59
BeefStroganoff	72.74	73.73	69.49	0.54	0.47
CarrotCake	75.34	78.93	68.90	0.58	0.39
CheeseBurger	72.74	73.89	68.15	0.47	0.45
ChickenSalad	68.99	71.27	62.39	0.53	0.50
ChickenStirFry	66.10	67.53	62.82	0.46	0.32
Coleslaw	70.05	74.06	77.59	0.51	0.64
CornChowder	77.97	80.23	72.19	0.62	0.45
DeviledEggs	88.57	89.13	60.66	0.81	0.36
EggNoodles	70.36	72.72	65.22	0.54	0.43
FrenchToast	68.95	73.43	72.76	0.58	0.53
MacAndCheese	78.73	80.36	68.78	0.64	0.45
MeatLasagna	73.25	74.44	65.23	0.55	0.44
PecanPie	63.91	64.37	60.44	0.48	0.39
PotatoSalad	79.30	80.30	64.12	0.68	0.39
PulledPork	68.48	72.73	70.09	0.50	0.45
PumpkinPie	66.60	67.68	58.45	0.55	0.51
VeggiePizza	69.43	72.95	64.11	0.39	0.40

Table 5.2: Sentence Re-ordering experiment

5.2.2 Global inference using viterbi sequence decoder : Content Model and Linear SVM Probability

We use viterbi style algorithm outlined in Algorithm 2 to identify the document level ordering of sentences. We use kendall tau [21] metric to compare the ordering identified by different approaches.

Refer Table 5.2 to compare the different approaches. Method 1 uses prediction probabilities directly from Linear kernel SVM to recover the document level ordering via our viterbi style algorithm. Similarly Method 2 uses probabilities outlined in Table 4.3 for viterbi decoding of document structure. Table 5.2 clearly shows that the SVM baseline is tough to beat for our content models. There is scope for modelling the interactions between arguments and

Method	Recall@10	Recall@30	Recall@50	Accuracy
Coref + 5 iterations	0.60	0.80	0.90	0.15
Coref + Text Order	0.45	0.80	0.85	0.05
5 Iterations without Coref	0.40	0.75	0.90	0.0

Table 5.3: Narrative Cloze for Recipe Events

actions in a recipe in a different way. For example, we could model the interactions via a HMM and see if we get better results for this experiment. We leave this as future work.

5.3 Narrative Event Cloze

We evaluate our content models by performing cloze test on the learnt event chains. From the graphs in the test-set we identify the longest chain per graph for evaluation. We leave one predicate in the chain at random and generate a list of ranked guesses using our content models. The search space for this experiment is the list of all cooking verbs that we have in our corpus. Similar in spirit to [7] and [8] we report the Recall at N metric. Our experiment is similar to the single protagonist model explained in [7]. Accuracy is measured based on the first guess returned by our models. If the first guess is the left out verb, the guess gets scored, otherwise not. For identifying the most probable candidate, we use the following scoring function in which $\prec$ indicates precedence relation:

$$score(a_{candidate}) = \sum_{1<i<candidate} log(P(a_i \prec a_{candidate})) + \sum_{candidate<i<n} log(P(a_{candidate} \prec a_i)) \tag{5.1}$$

From the Table 5.3 it is evident that iterative learning with co-reference resolution heuristics seem to outperform 2 other baselines.

Chapter 6

Conclusion and Future work

6.1 Limitations of Arborescence formulation

One of the limitations of our diagramming formulation is the fact that they could introduce parallel evolution edges, which is against our initial assumption that output of one action becomes either the input of another action either directly (through implicit argument edge) or indirectly (through evolution edge). For instance, refer Figure 6.1 to see a hypothetical case when the arborescence formulation may not guarantee a recipe flow graph consistent with our assumptions. We call this a theoretical limitation since we did not notice any such case empirically in our dataset.

6.2 Introduction to the GWAP

Like mentioned in Chapter 5, we need annotations done on a large scale to evaluate an activity diagramming algorithm. [22] introduced a game to collect image descriptions. The concept called Game with a Purpose (GWAP) has become popular since then and it has been used widely in several projects to collect human annotations on a large scale. [23], [24], [25] and [26] are some of the examples. In similar vein we introduce a GWAP to collect annotations for the recipes corpus.

The following sections in this chapter describe the Game With a Purpose (henceforth *GWAP*) for crowd sourcing annotations for the recipes corpus. There are 2 parts to the game. First part tries to collect labels for intra-sentence arguments and the second part for inter-sentence arguments. The second part of the game has 2 sections to it. One section of this part of the game tries to collect annotations for entities (ingredients or intermediate results in a recipe) involved in a referring expression (result or intermediate

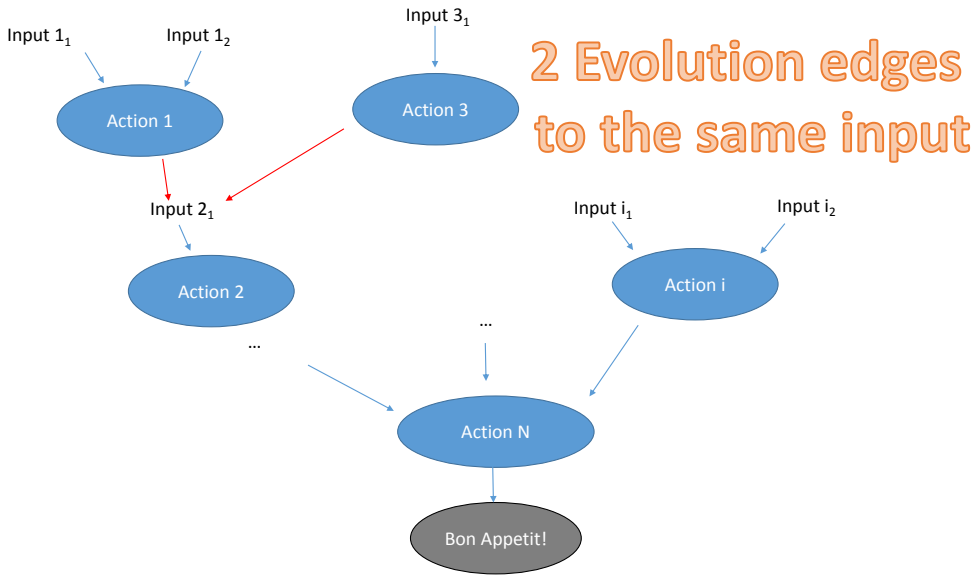

Figure 6.1: Hypothetical Limitation of Arborescence formulation

result in a recipe); the other section is for identifying implicit arguments for the cook predicates. As we will see, the case of implicit arguments becomes interesting when the implicit argument is not available in the immediately preceding sentence.

6.2.1 Intra-sentence Arguments

This part is intended for getting cooking predicates and their intra-sentence arguments. For prototyping we used openNLP chunker to identify different chunks of a sentence. A player will be presented with *(predicate, chunk-of-text)* pair. Then we will ask the player if the given *chunk-of-text* is object or co-object or location or duration or other of the *predicate*. (this is annotation mode of the game). Player has to say a simple yes or no.

For each question, a player will be shown definition of the label that is shown to him/her. For eg: for marking 'object' label, definition and examples for 'object' label will be shown in the UI; a snapshot of the UI is show in Figure 6.3.

In case the answer chosen by the player is 'no', we will ask the player to correct the label. The player will be rewarded based on the inter-annotator agreement, which would become useful as more number of people participate in the annotation. (validation mode for the annotations)

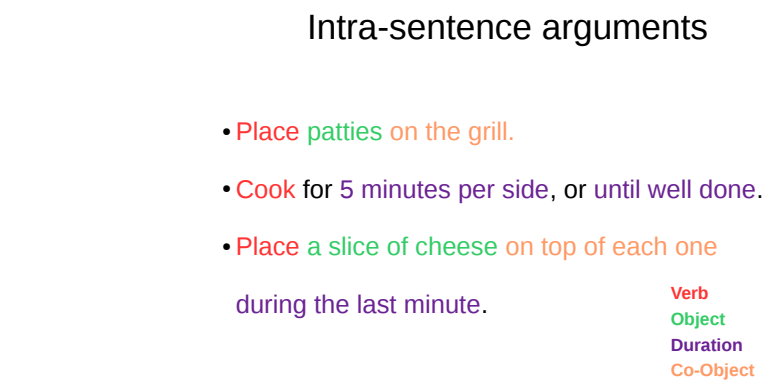

Figure 6.2: Examples for argument marking

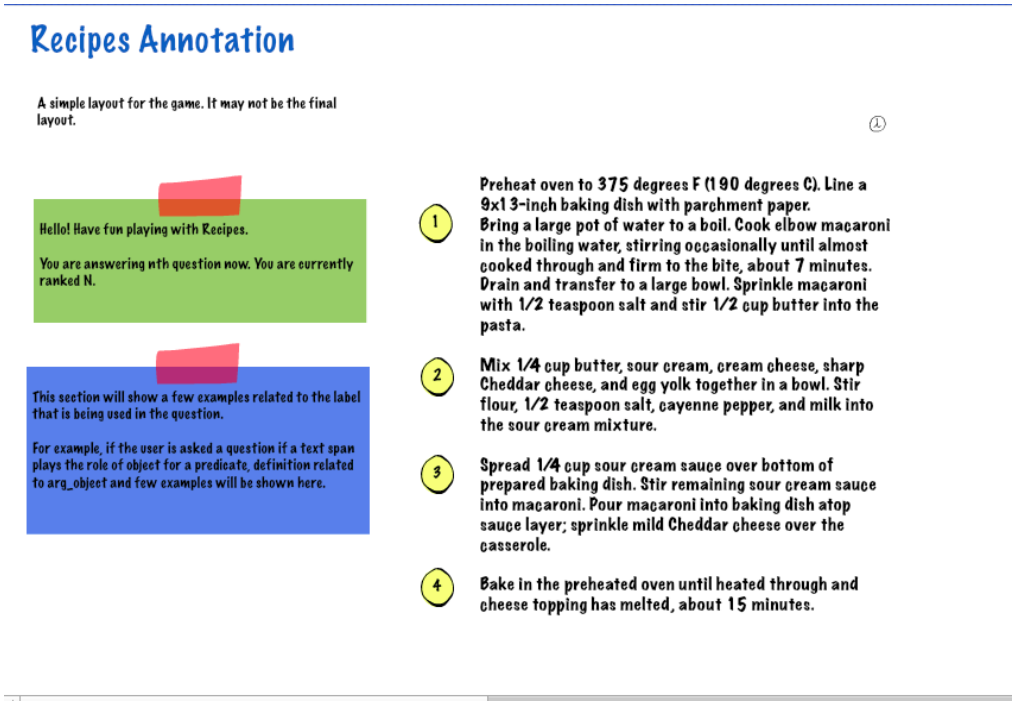

Figure 6.3: A sample layout for the game

To begin with, we will use some human-made annotations as bootstrap data. Once we have more annotations collected using the game, the annotations with inter-annotator agreements will become a part of the seed data.

To generate more questions, we will generate random labels for the chosen pair of *(predicate, chunk)* and ask the player if he/she agrees with the chosen

label. Like before, if a player disagrees with the label, we will ask the player to correct the label (or say 'Does not fit any category'), using his/her judgement.

6.2.2 Inter-sentence Arguments

Evolving Entities

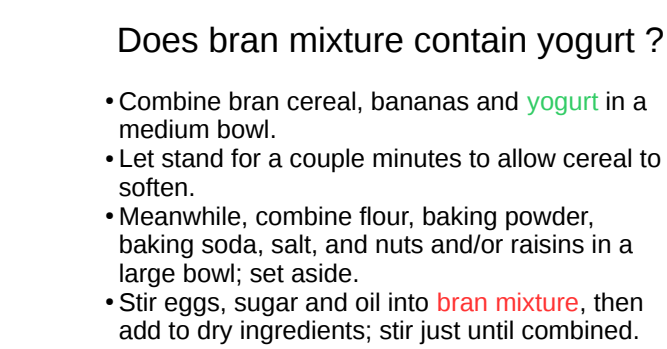

Figure 6.4: What's in the 'bran mixture'

One unique aspect of our corpus that we could capture in this game is the marking of evolving entities and what they refer to. For this part, we will identify a noun phrase $NP_{\text{Following}}$ (or human labelled evolved entities) and a random noun phrase that precedes this identified definite noun phrase $NP_{\text{Preceding}}$. For this pair $(NP_{Following}, NP_{Preceding})$ we ask the player to answer if $NP_{\text{Preceding}}$ is one of the ingredients/utensil in making $NP_{\text{Following}}$.

To bootstrap we could use some human labelled evolved-entities annotation. Once we have more players, response of one player could be used to question the other player's judgement thereby allowing us to collect inter-annotator agreement.

Figure 6.4 is a sample question given to the player. By looking at the text the player is expected to spot the fact that *bran and yogurt are combined to form the bran mixture.*

Figure 6.5 is an example of a wrong annotation given to the player. Player can mark the annotation as valid or invalid. If the player marks it as invalid, he/she will be asked to pick at least one ingredient that is a part of the expression 'dry ingredients'.

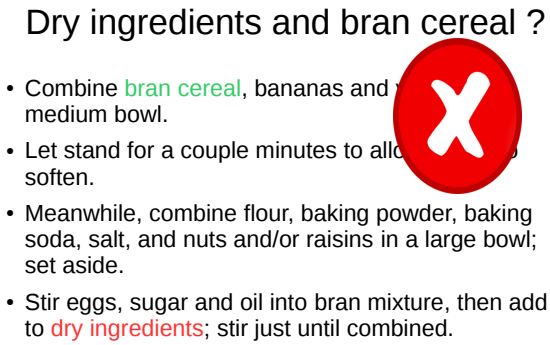

Figure 6.5: 'bran cereal' is not referred by 'dry ingredients'. 'bran mixture' in the text is the additional cue to the player

Implicit Arguments

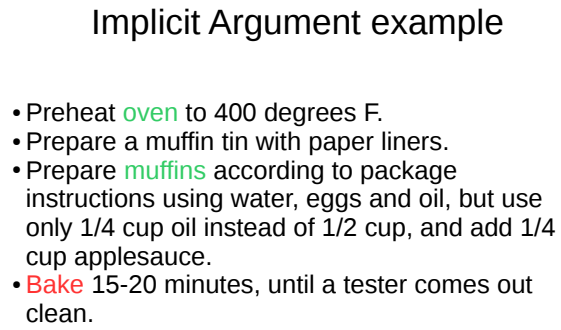

Figure 6.6: 'Bake' predicate with 2 implicit arguments

Similar to the Section 6.2.1, we give the players a *(predicate, chunk-of-text)* pair; except that the *chunk-of-text* is from any of the sentence preceding the sentence of the *predicate*. We expect to see a lot of negative labels here. Restricting ourselves only to the preceding sentence may not be a good idea as there are cases of implicit arguments that are far apart from the sentence containing the predicate. For example: In Figure 6.7, 'bake' predicate has 2 implicits arguments 'oven' and 'muffins' as highlighted.

Identifying suitable NPs for Inter-sentence markables

From the chunker output it is easy to identify NP chunks. However, we need to avoid NPs that are part of PPs that function as adverbial. To this end we use a set of heuristics to identify the PP as playing a role of adjective or adverb and filter for NPs that are part of adjective PPs, along with stand-alone NPs.

Upon simple post-processing we get a list of candidate markables that can participate in question generation for Section 6.2.2 and Section 6.2.2.

Find whether a PP is adverbial or adjective
Solution:
Heuristic 1: If the PP is first chunk in a sentence or is immediately preceded by VP, then it is more likely to be an adverbial.
Heuristic 2: If the PP is immediately preceded by a NP, it is more likely to be adjective PP.

The suitability of the above mentioned heuristics needs to be investigated for our All-Recipes corpus. Though, we don't have empirical evidence to support these heuristics, we believe this should be a good starting point for building the game.

6.2.3 Suitability for Activity Diagramming

It is worth mentioning that by having these 2 strategies (Section 6.2.2 and Section 6.2.2) for inter-sentence argument annotations, we will be able to retrieve the details required for evaluating our activity diagramming by simple post-processing. Also, we will capture the text-spans in the game to aid us in scoring of automatically generated diagrams, by using partial overlap of text-spans.

6.2.4 System Design

We present here a simple system design that could form the back-end for this proposed game. The design doesnt talk about handling user-login details now, but this should be a good place to start.

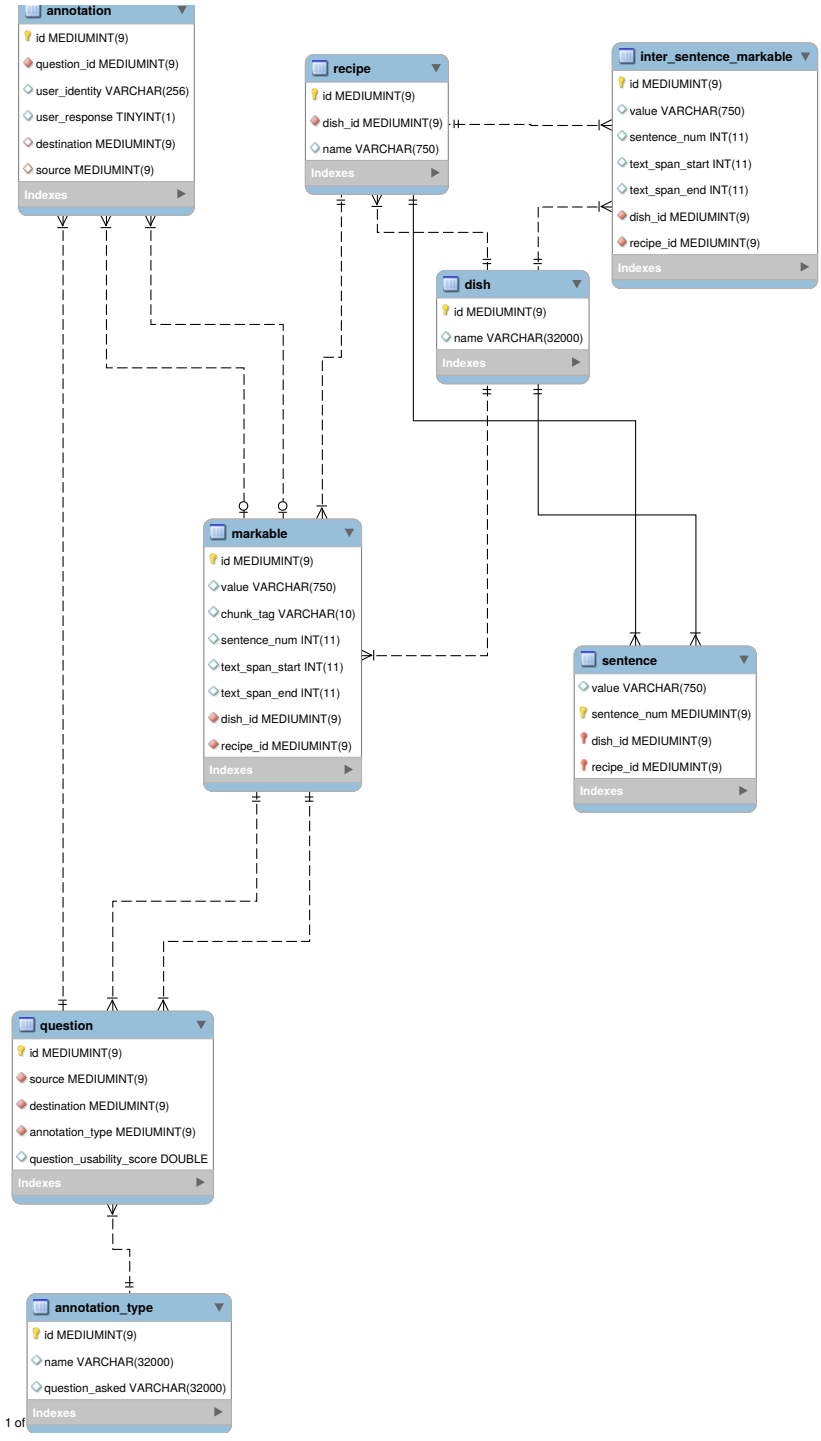

Figure 6.7: Entity Diagram for the simple game design

Bibliography

[1] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *ACL/HLT*, 2008. URL `pubs/narrative-chains08.pdf`.

[2] Matt Gerber, Joyce Y. Chai, and Adam Meyers. The role of implicit argumentation in nominal srl. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 146–154, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-41-1. URL `http://dl.acm.org/citation.cfm?id=1620754.1620776`.

[3] Charles J. Fillmore and Collin F. Baker. Frame semantics for text understanding. In *Proceedings of WordNet and Other Lexical Resources Workshop*, Pittsburgh, June 2001. NAACL, NAACL.

[4] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of EMNLP 2010*, 2010.

[5] B. A. Fox. Discourse structure and anaphora: written and conversational english. *Cambridge University Press*, 1993.

[6] Heeyoung Lee, Marta Recasens, Angel X. Chang, Mihai Surdeanu, and Dan Jurafsky. Joint entity and event coreference resolution across documents. In *EMNLP-CoNLL*, pages 489–500. ACL, 2012. ISBN 978-1-937284-43-5.

[7] Karl Pichotta and Raymond J. Mooney. Statistical script learning with multi-argument events. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014)*, Gothenburg, Sweden, April 2014. URL `http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127421`.

[8] Bram Jans, Steven Bethard, Ivan Vulic, and Marie-Francine Moens. Skip n-grams and ranking functions for predicting script events. In Walter Daelemans, Mirella Lapata, and Lluís Màrquez, editors, *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*, pages 336–344. The Association for Computer Linguistics, 2012. ISBN 978-1-937284-19-0. URL `http://aclweb.org/anthology-new/E/E12/E12-1034.pdf`.

[9] Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. Flow graph corpus from recipe texts. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asunción Moreno, Jan Odijk, and Stelios Piperidis, editors, *LREC*, pages 2370–2377. European Language Resources Association (ELRA), 2014.

[10] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[11] Paul Kingsbury and Martha Palmer. From treebank to propbank. In *LREC*. European Language Resources Association, 2002.

[12] Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the 2006 conference on Language Resources and Evaluation*, page 22312234, 2006.

[13] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In Erhard W. Hinrichs and Dan Roth, editors, *ACL*, pages 423–430. ACL, 2003.

[14] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, pages 1396–1400, 1965.

[15] J. Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards, 71B*, pages 233–240, 1967.

[16] Nikiforos Karamanis. Evaluating centering for sentence ordering in two new domains. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 65–68, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1614049.1614066`.

[17] Mirella Lapata. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 545–552, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075165. URL http://dx.doi.org/10.3115/1075096.1075165.

[18] Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225, 1995.

[19] Regina Barzilay and Mirella Lapata. Modeling local coherence: An entity-based approach. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 141–148, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219858. URL http://dx.doi.org/10.3115/1219840.1219858.

[20] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27, 2011. doi: 10.1145/1961189.1961199. URL http://doi.acm.org/10.1145/1961189.1961199.

[21] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2): pp. 81–93, 1938. ISSN 00063444. URL http://www.jstor.org/stable/2332226.

[22] Luis von Ahn and Laura Dabbish. ESP: labeling images with a computer game. In *Knowledge Collection from Volunteer Contributors, Papers from the 2005 AAAI Spring Symposium, Technical Report SS-05-03, Stanford, California, USA, March 21-23, 2005*, pages 91–98. AAAI, 2005. URL http://www.aaai.org/Library/Symposia/Spring/2005/ss05-03-014.php.

[23] Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In Grinter et al. [27], pages 55–64. ISBN 1-59593-372-7. doi: 10.1145/1124772.1124782. URL http://doi.acm.org/10.1145/1124772.1124782.

[24] Luis von Ahn, Shiry Ginosar, Mihir Kedia, Ruoran Liu, and Manuel Blum. Improving accessibility of the web with a computer game. In Grinter et al. [27], pages 79–82. ISBN 1-59593-372-7. doi: 10.1145/1124772.1124785. URL http://doi.acm.org/10.1145/1124772.1124785.

[25] Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *Computer Human Interaction*, pages 55–64, 2006. doi: 10.1145/1124772.1124782.

[26] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara L. Berg. Referit game: Referring to objects in photographs of natural scenes. In *EMNLP*, 2014.

[27] Rebecca E. Grinter, Tom Rodden, Paul M. Aoki, Edward Cutrell, Robin Jeffries, and Gary M. Olson, editors. *Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, 2006. ACM. ISBN 1-59593-372-7.