Understanding Restaurant Stories
Using an ASP Theory of Intentions

Qinglin Zhang and Daniela Inclezan

Miami University, Oxford OH 45056, USA
`zhangq7,inclezd@miamioh.edu`

Abstract. This paper describes an application of logic programming to the understanding of narratives. Substantial work in this direction has been done by Erik Mueller, who focused on texts about *stereotypical activities* or *scripts*, in particular restaurant stories. His system was able to demonstrate a deep understanding of a good number of such narratives. However, texts describing exceptional scenarios could not be processed correctly by his system. We propose addressing this problem by using an Answer Set Prolog theory of intentions developed by Blount, Gelfond, and Balduccini. We present a methodology in which we model scripts as *activities* and employ the concept of an *intentional agent* to reason about both normal and exceptional scenarios. We exemplify the application of this methodology on a number of restaurant stories.

1 Introduction

This paper describes an application of logic programming to the understanding of narratives. According to Schank and Abelson [13], stories frequently narrate episodes related to *stereotypical activities — sequences of actions normally performed in a certain order by one or more actors, according to cultural conventions.* One example of a stereotypical activity is dining in a restaurant, in which the following actions are expected to occur: the customer enters, he is greeted by the waiter who leads him to a table, the customer sits down, reads the menu, orders some dish, the waiter brings the dish, the customer eats and then asks for the bill, the waiter places the bill on the table, the customer pays and then leaves. A story mentioning a stereotypical activity is not required to state explicitly all of the actions that are part of it, as it is assumed that the reader is capable of filling in the blanks with his own commonsense knowledge about the activity [13]. Consider, for instance, the following narrative:

Example 1 (Scenario 1, from[12]). Nicole went to a vegetarian restaurant. She ordered lentil soup. The waitress set the soup in the middle of the table. Nicole enjoyed the soup. She left the restaurant.

Cultural norms indicate that customers do not seat themselves in a restaurant, but rather wait to be seated by the waiter. Similarly, customers are expected to pay for their food. As readers are supposed to know these conventions, such information is missing from the text.

Schank and Abelson [13] introduced the concept of a *script* to model stereotypical activities: a *fixed* sequence of actions that are *always* executed in a specific order. Following these ideas, Erik Mueller conducted substantial work on narratives about stereotypical activities. He focused on restaurant stories in [12] and news about terrorist incidents in [11]. In [12], Mueller developed a system that can take as an input a text about a restaurant episode, process it using information extraction techniques, and demonstrate a deep understanding of the narrative by answering questions whose answers are not necessarily explicitly stated in the text. A commonsense knowledge base about the restaurant domain was used in the question answering task. The system had a good accuracy but the rigidity of scripts did not allow for the correct processing of scenarios describing variations (e.g., several dishes being ordered) nor exceptions (e.g., waiter bringing a wrong dish). To be able to handle such scenarios, all variations and possible exceptions of a script would have to be foreseen and encoded as new scripts by the designer of the knowledge base, which is an important hurdle. *In this paper, we concentrate on designing a more flexible methodology suitable for both normal and exceptional scenarios.* We ignore script variations for now, as well as the natural language processing task that is orthogonal to our goal.

We propose to view the main actor in a stereotypical activity (the customer in a restaurant scenario), as an agent that *intends* to perform some actions in order to achieve a goal, but may not always need to/ be able to perform them as soon as intended. It is instrumental for our purpose to use a *theory of intentions* developed by Blount *et al.* [5,6] that introduces the concept of an *activity* — a sequence of agent actions and sub-activities that are supposed to achieve a goal. Blount *et al.*'s theory of intentions is written in an action language and is translatable into Answer Set Prolog (ASP) [9]. It can be easily coupled with ASP commonsense knowledge bases about actions and their effects, and with reasoning algorithms encoded in ASP or its extensions, to build executable systems. We are not aware of any other logic programming theories of intentions that can be immediately integrated in executable systems.

Blount *et al.*'s also introduce an architecture ($\mathcal{AIA}$) of an *intentional agent*, (an agent that obeys his intentions). According to $\mathcal{AIA}$, at each time step, the agent observes the world, explains observations incompatible with its expectations (diagnosis), and determines what action to execute next (planning). The reasoning module implementing $\mathcal{AIA}$ allows an agent to reason about a wide variety of scenarios, including the serendipitous achievement of its goal by exogenous actions or the realization that an active activity has no chance to achieve its goal anymore (i.e., it is futile). We see possible applications of (parts of) this reasoning module to the following exceptional restaurant scenarios.

Example 2 (Serendipity). Nicole went to a vegetarian restaurant. She ordered lentil soup. When she waitress brought her the soup, she told her that it was on the house. (The reader should understand that Nicole did not pay for the soup.)

Example 3 (Detecting Futile Activity). Nicole went to a vegetarian restaurant. She sat down and wanted to order lentil soup, but it was not on the menu. (The reader should deduce that Nicole stopped her plan of eating lentil soup here.)

Example 4 (Diagnosis). Nicole went to a vegetarian restaurant. She ordered lentil soup. The waitress brought her a miso soup instead. (The reader is supposed to produce some explanations for what may have gone wrong: either the waitress or the cook misunderstood the order.)

In contrast with $\mathcal{AIA}$, which encodes an agent's reasoning process about his own goals, intentions, and ways to achieve them, we need to represent the reasoning process of a (cautious) reader that learns about the actions of an intentional agent from a narrative. For instance, while an intelligent agent creates or selects its own activity to achieve a goal, in a narrative context, the reader learns about the activity that was selected by the agent from the text. As a consequence, *one of our goals in this work is to understand what parts of the $\mathcal{AIA}$ architecture can be adapted to our purposes of modeling the reasoning of a narrative reader and how.* There were two main difficulties here. First, narratives (especially the ones about stereotypical activities) do not state explicitly all of the actions that happen. As a result, there are fewer time points on the story time line than on the reasoning time line, and a matching between the two has to be found. Second, stereotypical activities normally include several agents, not just one. In a dining episode, other than the customer that can be considered the main character, there is a waiter, a cook, and possibly other customers. To reduce the complexity of reasoning about multiple intentional agents for now, we used an older and simpler theory of intentions by Baral and Gelfond [4] to model the intended sequences of actions for secondary characters (e.g., waiter and cook).

The methodology we describe here can be applied to narratives about other stereotypical activities. This is just a first exploration of the subject, which can be further expanded in the future. The remainder of the paper provides some review of related and background work. It then continues with the description of the methodology and its evaluation on some sample scenarios. It ends with a discussion of future work and conclusions.

2 Related Work

Restaurant Narratives. Erik Mueller's work is based on the hypothesis that readers of a text understand it by constructing a mental model of the narrative. An extensive review of narrative processing systems can be found in [12]. Mueller's system described in [12] showed a deep understanding of restaurant narratives by answering questions about time and space aspects that were not necessarily mentioned explicitly in the text. His system relied on two important pieces of background knowledge: (1) a commonsense knowledge base about actions occurring in a restaurant, their effects and preconditions, encoded in Event Calculus [14] and (2) a script describing a sequence of actions performed by different characters in a *normal* unfolding of a restaurant episode.

The system processed English text using information extraction techniques in order to fill out slot values in a template. For the scenario in Example 1, for instance, the slot CUSTOMER would be filled with value "Nicole" and SCRIPT:

LAST EVENT with Leave. Possible values for the latter included Arrive, Sit, ReadMenu, Order, Serve, Eat, RequestBill, ReceiveBill, Pay, and Leave, corresponding to actions in the restaurant script. Next, the template was translated into a reasoning problem that contained: facts about the entities identified in the template; facts about the consecutive occurrence of all actions in the script up to the one corresponding to the value of the SCRIPT: LAST EVENT slot; and default information about the layout of the restaurant and the locations of different objects and characters. Then, the reasoning problem was expanded with the information in the commonsense knowledge base to compute models of the input restaurant scenario. Finally, questions about time and space aspects were automatically generated and answers were obtained from the model resulting in the previous step. Mueller's system was tested on roughly 200 excerpts of texts retrieved from the internet or Project Gutenberg collection, and answered correctly 70% of the test questions. In terms of limitations of the system, the author cited the lack of flexibility of scripts, which resulted in scenarios with exceptional cases (or variations) not being processed correctly.

Activity Recognition. The task we undertake here is somewhat connected to activity recognition, in that it requires observing agents and their environment in order to complete the picture about the agents' actions and activities. However, unlike activity recognition, understanding narratives about stereotypical activities does not require identifying an agent's goal, which is always the same in our case. Gabaldon [7] performed activity recognition using an simpler theory of intentions by Baral and Gelfond [4] that did not consider goal-driven agents.

3 Preliminary: Theory of Intentions

Old Theory of Intentions by Baral and Gelfond

Baral and Gelfond [4] captured properties of intended actions in an ASP theory we denote by $old\mathcal{TI}$ that had two main tenets: *"Normally intended actions are executed the moment such execution becomes possible"* (non-procrastination) and *"Unfulfilled intentions persist"* (persistence). Sequences of actions were modeled using predicates of the type: $sequence(\mathbf{s})$ ($\mathbf{s}$ is a sequence); $length(\mathbf{n}, \mathbf{s})$ ($\mathbf{n}$ is the length of sequence $\mathbf{s}$); and $component(\mathbf{s}, \mathbf{k}, \mathbf{x})$ (the $\mathbf{k}^{th}$ element of sequence $\mathbf{s}$ is $\mathbf{x}$). An agent's intentions at different time points was captured by the predicate $intend(\mathbf{x}, \mathbf{i})$ (action/ sequence $\mathbf{x}$ is intended at time step $\mathbf{i}$). The theory was successfully used in activity recognition [7] and question answering [10], but was not sufficient for modeling goal-oriented agents.

New Theory of Intentions by Blount *et al.*

Blount *et al.* [5,6] improved on the previous theory of intentions by considering goal-driven agents. For this purpose, each sequence of actions of an agent was associated with a goal that it was meant to achieve – the combination of the two was now called an *activity*. Activities could have nested sub-activities, and were encoded using the predicates: $activity(\mathbf{m})$ ($\mathbf{m}$ is an activity); $goal(\mathbf{m}, \mathbf{g})$ (the goal of activity $\mathbf{m}$ is $\mathbf{g}$); $length(\mathbf{n}, \mathbf{m})$ (the length of activity $\mathbf{m}$ is $\mathbf{n}$); and

$component(\mathbf{m}, \mathbf{k}, \mathbf{x})$ (the $\mathbf{k}^{th}$ component of activity $\mathbf{m}$ is $\mathbf{x}$, where $\mathbf{x}$ is either an action or a sub-activity).

The authors introduced the concept of an *intentional agent* — one that has goals that it intends to pursue, "only attempts to perform those actions that are intended and does so without delay." As normally done in our field, the agent is expected to possess knowledge about the changing world around it. This can be represented as a transition diagram in which nodes denote *physical* states of the world and arcs are labeled by *physically executable* actions that may take the world from one state to the other. States describe the values of relevant properties of the world, where properties are divided into fluents (those that can be changed by actions) and statics (those that cannot). To accommodate intentions and decisions of an intentional agent, Blount *et al.* expanded the traditional transition diagram with *mental* fluents and actions. Two important mental fluents in $new\mathcal{TI}$ are $status(\mathbf{m}, \mathbf{k})$ ($\mathbf{m}$ is in progress if $\mathbf{k} \geq 0$, and not yet started or stopped if $\mathbf{k} = -1$) and $next\_action(\mathbf{m}, \mathbf{a})$ (the next action to be executed as part of activity $\mathbf{m}$ is $\mathbf{a}$). Mental actions included $select(\mathbf{g})$ and $abandon(\mathbf{g})$ for goals, and $start(\mathbf{a})$ and $stop(\mathbf{a})$ for activities. The new transition diagram was encoded in an action language; we denote by $new\mathcal{TI}$ its ASP translation.

Additionally, Blount *et al.* developed an agent architecture $\mathcal{AIA}$ (implemented in CR-Prolog [2,1], an extension of ASP) that adapts the agent loop [3] to accommodate an *intentional* agent. For instance, while fluent $next\_action(\mathbf{m}, \mathbf{a})$ in the theory of intentions indicates the action in activity $\mathbf{m}$ that the agent would *normally* need to execute next, the agent architecture handles exceptions to this rule. The decision not to execute the next action is made if the activity's goal was already achieved by some other action (Example 2); if the goal of the current activity was abandoned; or if the current activity needs to be stopped altogether because it no longer has chances of achieving its goal (Example 3).

4 Methodology

In this section, we outline a methodology of using theories of intentions and parts of the $\mathcal{AIA}$ architecture to design a program that can show a deep understanding of stories about stereotypical activities, exemplified on restaurant stories. We ignore the natural language processing part, a difficult task on it own, and focus on the reasoning component only. We distinguish between the story time line containing strictly the events mentioned in the text and the reasoning time line corresponding to the mental model that the reader constructs. We begin with assumptions and the general steps; we then discuss each step in more detail.

Assumptions. We assume that a wide coverage commonsense knowledge base ($c\mathcal{KB}$) written in ASP is available to us and that it contains information about a large number of actions, their effects and preconditions, including actions in the stereotypical activity. How to actually build such a knowledge base is a difficult research question, but it is orthogonal to our goal. In practice, in order to be able to evaluate our methodology, we have built a basic knowledge base with core information about restaurants and, whenever a scenario needed new

information, we expanded the knowledge base with new actions and fluents. We operated under the assumption that all this information would be available from the beginning in the commonsense knowledge base $c\mathcal{KB}$.

Since this is the first attempt to use a theory of intentions to reason about stereotypical activities, including exceptional cases, we also consider the following simplifying assumptions: only one main character (the customer that wants to dine) and exactly one character per every other important role (i.e., one waiter and one cook), only one ordered dish.

Steps in Our Methodology. According to our methodology, for each input text t, we construct a logic program $\Pi(t)$ whose answer sets represent models of the narrative (useful for question answering). This logic program has two parts, one that is pre-defined, and another one that depends on the input text.

The **pre-defined part** consists of the following items:

1. The $c\mathcal{KB}$ knowledge base, with a core describing sorts, fluents, actions, and some pre-defined objects relevant to the stereotypical activity of focus;
2. The two theories of intentions, $old\mathcal{TI}$ and $new\mathcal{TI}$;
3. A module encoding the stereotypical activity as an activity of $new\mathcal{TI}$ for the main character and as sequences of $old\mathcal{TI}$ for the remaining characters; and
4. A reasoning module, encoding (i) a mapping of time points on the story time line into points on the reasoning time line; (ii) reasoning components adapted from the $\mathcal{AIA}$ architecture to reflect a reader's reasoning process and expected to allow reasoning about serendipitous achievement of goals, decisions to stop futile activities, and diagnosis.

The **input-dependent part** (i.e., the logic form obtained by translating the English text into ASP facts) consists of the following:

5. Facts defining objects mentioned in the text as instances of relevant sorts in the $c\mathcal{KB}$ (including who the main character is, etc.)
6. Observations about the values of fluents and the occurrences of actions at different points on the *story* time line.
7. Default information about the values of fluents in the initial situation.

We now elaborate on the steps mentioned above.

4.1 The Core of the Commonsense Knowledge Base $c\mathcal{KB}$

The core of $c\mathcal{KB}$ defines knowledge related to the restaurant environment. It includes a hierarchy of sorts with two main sorts, *thing* and *location*; the former is divided into *person*, *food*, *restaurant*, and *bill* where *person* has sub-sorts *customer*, *waiter*, and *cook*. We consider some special sub-sorts: *the_customer*, *the_waiter*, *the_food*, and *the_restaurant*, denoting the main customer, the waiter, food, and restaurant in the text, respectively. Some pre-defined instances of sorts are also specified: `entrance`, `kitchen`, `counter`, `outside` are instances of *location*; `m` and `t` are *thing*s denoting the unique menu and table we are considering for now; `t` is also an instance of *location*; and `b` is the main customer's bill. The core describes actions and fluents related to the restaurant environment,

which can be seen in Figure 1 in which **t** denotes the table and we use **c** for a *customer*; **w** for a *waiter*; **ck** for a *cook*; **f** for a *food*; **r** for a *restaurant*; **t1** and **t2** for *thing*s; **l**, **l1** and **l2** for *location*s; **p**, **p1**, and **p2** for *person*s. Note that *interference* is an example of an exogenous action (not an agent action) that can be relevant for diagnostic (see Example 4). All fluents are inertial (i.e., they normally maintain their previous values unless changed by an action), while the last three, *ready_to_eat*(**c**, **f**), *done_with_payment*(**c**), and *satiated_and_out*(**c**), are defined (i.e., they are completely defined in terms of other fluents).

Actions		Fluents	
enter(**c**, **r**)	*request*(**p1**, **t**, **p2**)	*hungry*(**c**)	*informed*(**p1**, **t**, **p2**)
greet(**w**, **c**)	*prepare*(**ck**, **f**)	*open*(**r**)	*food_prepared*(**ck**, **f**)
move(**p**, **l1**, **l2**)	*eat*(**c**, **f**)	*at_loc*(**t**, **l**)	*bill_generated*(**c**)
lead_to(**w**, **c**, **t**)	*pay*(**c**)	*in*(**c**, **r**)	*paid*(**b**)
sit(**c**)	*stand_up*(**c**)	*welcomed*(**c**)	*available*(**f**, **r**)
pick_up(**p**, **t1**, **t2**)	*leave*(**c**)	*standing_by*(**p**, **l**)	*ready_to_eat*(**c**, **f**)
put_down(**p**, **t1**, **t2**)		*sitting*(**c**)	*done_with_payment*(**c**)
order(**c**, **f**, **w**)	*interference*	*holding*(**p**, **t**)	*satiated_and_out*(**c**)

Fig. 1. Important Actions and Fluents in the Restaurant-core of *c$\mathcal{KB}$*

Axioms about the direct, indirect effects and preconditions of actions are encoded in ASP using a standard methodology [8]. We show the more interesting encoding of the direct effects of action *interference*. Simultaneous occurrence of this action with *order*(**c**, **f**, **w**) or *request*(**p1**, **t**, **p2**) may cause miscommunication. We encode this as a non-deterministic direct effect via the rules below:

$$1\{holds(informed(W, F1, C), I + 1) : other\_food(F1, F)\}1 \leftarrow$$
$$occurs(order(C, F, W), I),$$
$$occurs(interference, I).$$
$$other\_food(F_1, F) \leftarrow food(F), food(F_1), F \neq F_1.$$
$$holds(informed(W, F, C), I + 1) \leftarrow occurs(order(C, F, W), I),$$
$$\neg occurs(interference, I).$$

The first axiom says that an *interference* occurring at the same time as a customer's action of ordering some food F causes the waiter to understand that the customer is asking for a different food than F. The third rule indicates that the waiter understands the customer's order correctly when there is no simultaneous *interference*. There are similar rules in *c$\mathcal{KB}$* for action *request*(**p1**, **t**, **p2**). Defined fluents like *satiated_and_out* are defined by rules such as:

$$holds(satiated\_and\_out(C), I) \leftarrow holds(satiated(C), I),$$
$$holds(at\_l(C, outside), I).$$

4.2   Encoding Stereotypical Activities

Narratives about stereotypical activities include multiple characters with their own goals and actions. For now, we reduce the complexity of the problem by considering that the reader only views the main character as a goal-oriented agent and assumes that all other characters are not (they just obey orders).

Thus, we use activities and the $new\mathcal{TI}$ for the main character (the customer in a dining episode), and sequences of actions and the $old\mathcal{TI}$ for other (i.e., the waiter and cook).

Customer Activities. We discovered that, instead of modeling the customer's stereotypical actions as a flat activity, it is better to introduce as many sub-activities with sub-goals as reasonable. This becomes useful when dealing with exceptional cases. We introduce an activity $c\_act(C, R, F)$ defined as

$activity(c\_act(C, R, F)) \leftarrow the\_customer(C), the\_restaurant(R), the\_food(F)$.

Its goal is for the customer to be satiated and outside the restaurant:

$goal(c\_act(C, R, F), satiated\_and\_out(C)) \leftarrow activity(c\_act(C, R, F))$.

We define the activity to be associated with the list of actions/ sub-activities $\langle\ enter(C, R),\ sit(C),\ c\_subact\_1(C, F),\ eat(C, F),\ c\_subact\_2(C),\ stand\_up(C),\ leave(C)\ \rangle$ This is done via rules of the type:

$component(c\_act(C, R, F), 1, enter(C, R)) \leftarrow activity(c\_act(C, R, F))$.
$length(7, c\_act(C, R, F)) \leftarrow activity(c\_act(C, R, F))$.

Sub-activities $c\_subact\_1(C, F)$ and $c\_subact\_2(C)$ are defined as having the goals $ready\_to\_eat(C, F)$ and $done\_with\_payment(C)$, resp., and containing the lists $\langle\ pick\_up(C, \mathtt{m}, \mathtt{t}),\ put\_down(C, \mathtt{m}, \mathtt{t}),\ order(C, F, W)\ \rangle$ and $\langle\ request(C, \mathtt{b}, W),\ pay(C, \mathtt{b})\ \rangle$, respectively (where $\mathtt{m}$ is the menu, $\mathtt{t}$ is the table, and $\mathtt{b}$ is the bill).

Waiter Sequences. We describe the sequence of actions performed by a waiter using the $old\mathcal{TI}$. We introduce objects of the form $w\_seq(W, C, F1, F2)$ where W is the waiter, C is the customer, and $F1$ and $F2$ are foods mentioned in the scenario, but not necessarily the food ordered by the customer. The two food parameters allow us to reason about exceptional scenarios like Example 4; in such cases multiple sequences are generated for a waiter, but the waiter only intends one of them. A sequence $w\_seq(W, C, F1, F2)$ consists of the following actions: $\langle\ greet(W, C),\ lead\_to(W, C, \mathtt{t}),\ move(W, \mathtt{t}, \mathtt{kitchen}),\ request(W, F1, \mathtt{ck}),\ pick\_up(W, F2, \mathtt{kitchen}),\ move(W, \mathtt{kitchen}, \mathtt{t}),\ put\_down(W, F2, \mathtt{t}),\ move(W, \mathtt{t}, \mathtt{counter}),\ pick\_up(W, \mathtt{b}, \mathtt{counter}),\ move(W, \mathtt{counter}, \mathtt{t}),\ put\_down(W, \mathtt{b}, \mathtt{t})\rangle$.

Cook Sequences. Similarly, we define the $old\mathcal{TI}$ sequences of actions performable by a cook as being of the form $ck\_seq(Ck, F, W)$ where Ck is a cook, F is a food (not necessarily the food ordered by the customer), and W is a waiter. A cook sequence $ck\_seq(Ck, F, W)$ consists of a single action: $\langle prepare(Ck, F, W)\rangle$. Since we use sequences for both the waiter and the cook, we use the predicate $actor(\mathbf{s}, \mathbf{ac})$ to specify that sequence $\mathbf{s}$ is performed by $\mathbf{ac}$.

Default Information about Restaurant Scenarios. In relation to the customer's activity and the waiter's and cook's sequences, we also define some default facts that complement the information that can be extracted from a narrative. These state that, in any restaurant scenario, the customer initially (i.e., at time step 0 on the reasoning time line) selects the goal of being $satiated\_and\_out$; he starts his activity c_act at the next time step. This is encoded as follows, using the methodology for recoding the occurrence of actions outlined in [6]:

$hpd(select(satiated\_and\_out(C)), true, 0) \leftarrow the\_customer(C)$.
$hpd(start(c\_act(C, R, F)), true, 1) \leftarrow activity(c\_act(C, R, F))$.

In a normal scenario, there would be only one possible sequence for the waiter and one for the cook, which would be intended at time step 0. However, in exceptional scenarios, there may be several such sequences covering possible mistakes made by the waiter or cook. Thus, we say that the waiter intends to perform *one of its possible sequences* at time step 0 (similarly for the cook):

$1\{intend(S, 0) : actor(S, W)\}1 \leftarrow the\_waiter(W).$

$1\{intend(S, 0) : actor(S, Ck)\}1 \leftarrow the\_cook(Ck).$

4.3 Reasoning Module

We now concentrate on the reasoning module of the $\mathcal{AIA}$ architecture associated with $new\mathcal{TI}$ and determine what parts of it can be imported/ adapted and what new rules need to be added in order to capture the reasoning process of a reader of a narrative. In what follows, we denote the reader's reasoning module by RM and start with a couple of key points in its $\mathcal{AIA}$-inspired construction:

● A reader needs to map observations about fluents and actions (i.e., a *history*) into the predicates *holds* and *occurs* used for reasoning. It also needs to perform diagnostic (just like an intentional agent would) when reading pieces of information that are unexpected, such as a waiter bringing the wrong dish. Thus *the temporal projection and diagnostic modules of $\mathcal{AIA}$ are imported into RM.*

● The reader needs to fill the story time line with new time points (and thus construct what we call a *reasoning time line*) to accommodate mental and physical actions not mentioned in the text. *This is achieved by adding new rules to RM*, in which we denote story vs. reasoning time steps by predicates *story_step* and *step*, respectively, and introduce predicate $map(\mathbf{ss}, \mathbf{i})$ to say that story step $\mathbf{ss}$ is mapped into reasoning time step $\mathbf{i}$:

$1\{map(SS, I) : step(I)\}1 \leftarrow story\_step(I).$

$\neg map(SS_1, I_1) \leftarrow map(SS, I), SS < SS_1, I \geq I_1, story\_step(S), step(I).$

Observations about the occurrence of actions and values of fluents recorded from the text using predicates *st_hpd* and *st_obs* as described in Subsection 4.4 are translated into observations on the reasoning time line via rules of the type:

$hpd(A, true, I) \leftarrow st\_hpd(A, true, SS), map(SS, I).$

Finally, we do not want to create time steps on the reasoning time line when no action is believed to have occurred (i.e., gaps). We encode this using the rules

$something\_occurs(I) \leftarrow occurs(A, I).$

$\leftarrow last\_assigned(I), step(J), J < I, not\ something\_occurs(J).$

where $last\_assigned(\mathbf{i})$ is true if $\mathbf{i}$ is the last time step on the reasoning time line that has a correspondent on the story time line.

Blount *et al.* considered four categories of histories and described the encoding of the corresponding agent strategies in $\mathcal{AIA}$. We now analyze each category separately and discuss its suitability for RM, as well as pertinent changes:

Category 1. *No goal nor activity to commit to. In $\mathcal{AIA}$, the agent waits.* In narratives about stereotypical activities, the main character has an active (predefined) goal and activity, so $\mathcal{AIA}$ category 1 histories are not relevant to RM.

Category 2. *A top-level activity is active but its goal is not. In $\mathcal{AIA}$, the agent stops the activity.* Serendipitous achievement of the main character's goal by someone else's actions can happen in narratives about stereotypical activities (see Example 2). Thus $\mathcal{AIA}$ histories of category 2 are relevant and the rules describing the associated agent strategy in $\mathcal{AIA}$ are included in RM.

Category 3. *A top-level activity and its goal are active. In $\mathcal{AIA}$, the agent performs the next action, unless there are no chances for the activity to still achieve its goal (i.e., the activity is deemed* futile*), in which case it is stopped.* RM imports the corresponding rules for the agent strategy, but changes the definition of a futile activity. For instance, if Nicole finds the restaurant closed, according to the $new\mathcal{TI}$ there is still a chance that it will open later, which means that Nicole's activity is not futile and she will persist with it. In real life however, Nicole may not wait more than 30 minutes to satisfy her goal of not being hungry. Given that we do not intend to explore real time aspects at the moment, we suggest adding some additional background knowledge to the module describing activities and sequences (see Subsection 4.2). The default information would state that an activity of type $c\_act(\mathbf{c}, \mathbf{r}, \mathbf{f})$ is futile if $\mathbf{r}$ is observed to be closed when $\mathbf{c}$ wants to enter; if $\mathbf{f}$ is observed to be unavailable at $\mathbf{r}$; etc., which we encode as:

$$futile(c\_act(C, R, F), I) \leftarrow obs(open(R), false, I).$$
$$futile(c\_act(C, R, F), I) \leftarrow obs(available(F, R), false, I).$$

More complex rules can also be added, such as that this activity is futile if no table is available and the customer is impatient.

Category 4. *A goal is active but there is no active activity to achieve it. The agent needs to find a plan (i.e., start a new activity).* Our cautious reader is not expected to guess or assume the new plan the main character computes or selects, but rather determine that re-planning is required. We introduce a new mental action $replan(\mathbf{g})$ where $\mathbf{g}$ is a goal, and add the following rule to RM:

$$occurs(replan(G), I) \leftarrow category\_4\_history(G, I).$$

This concludes the description of the main parts of RM and that of the pre-defined parts in the logic program constructed according to our methodology.

4.4 Logic Form

We now describe the input-dependent part of the logic program we construct for every narrative. We assume that a text is translated into a logic form that contains two parts: (a) definitions of instances of sorts *the_customer*, *the_waiter*, *the_food*, *food*, *the_restaurant*; and (b) observations about the values of fluents and occurrence of actions in relation to the *story* time line. For (a), if the name of one of these entities is not given in the text, then it is replaced by a new constant (e.g., cook1). For (b), in order to distinguish between the story time line and the reasoning time line, we substitute the predicates $obs(\mathbf{f}, \mathbf{v}, \mathbf{i})$ and $hpd(\mathbf{a}, \mathbf{v}, \mathbf{i})$ normally used in the description of histories [6] by $st\_obs(\mathbf{f}, \mathbf{v}, \mathbf{ss})$ (fluent $\mathbf{f}$ from the $c\mathcal{KB}$ has value $\mathbf{v}$ at time step $\mathbf{ss}$ in the story time line, where $\mathbf{v}$ may be true or false) and $st\_hpd(\mathbf{a}, \mathbf{v}, \mathbf{ss})$ (action $\mathbf{a}$ from the $c\mathcal{KB}$ was observed to have

occurred if **v** is `true`, or not if **v** is `false`, at time step **ss** in the story time line). In addition to the observations obtained directly from the text (mentioned there explicitly), we assume that the logic form will also contain *default* observations such as the fact that the restaurant is assumed to be open, the customer is hungry, the waiter is at the entrance, and so on.

Example 5 (Logic Form). The text in Example 1 would be translated into a logic form that includes the following facts in addition to the default observations:

the_customer(`nicole`).	*st_hpd*(*enter*(`nicole`, `veg_r`), `true`, 0).
the_restaurant(`veg_r`).	*st_hpd*(*order*(`nicole`, `lentil_soup`, `waitress`), `true`, 1).
the_food(`lentil_soup`).	*st_hpd*(*put_down*(`waitress`, `lentil_soup`, `t`), `true`, 2).
the_waitress(`waitress`).	*st_hpd*(*eat*(`nicole`, `lentil_soup`), `true`, 3).
the_cook(`cook1`).	*st_hpd*(*leave*(`nicole`), `true`, 4).

5 Evaluation

We applied our methodology to a collection of fifteen stories, describing both normal and exceptional scenarios like the ones in Examples 1-4. We show here the output for some of them and indicate how to interpret it.

Normal Scenario in Example 1. The answer set of the program $\Pi(1)$ obtained according to our methodology contains the *occurs*($\mathbf{a}$, $\mathbf{i}$) atoms shown in Appendix A, where $\mathbf{a}$ is an action and $\mathbf{i}$ is a time point on the reasoning time line. We use this case as a baseline when explaining the output of exceptional scenarios. In this output, note how the customer's actions are interleaved with those of the waitress and cook. As well, note that mental actions such as starting a sub-activity (occurring at time steps 6 an 18) or stopping one (steps 16 and 25) take one time step when no other action occurs. The first customer sub-activity stops only once the soup is on the table and thus the customer is *ready_to_eat*; the second one stops after Nicole pays for her bill and thus she is *done_with_payment*.

Serendipitous Achievement of Goal in Example 2. The logic form for this scenario is identical to the one for Example 1 shown in 5, except that the two observations about actions taking place at time points 2–4 are replaced by

st_hpd(*pay*(`owner`, `b`), `true`, 2). *st_hpd*(*put_down*(`waitress`, `lentil_soup`, `t`), `true`, 3).

where `owner` is a new instance of sort *people*. The answer set of $\Pi(2)$ contains similar *occurs* atoms to $\Pi(1)$ plus one for action *pay*(`owner`, `b`) up to time step 19 when the customer's sub-activity related to the bill payment *c_subact_2* starts. From then on, it contains the following *occurs* predicates:

occurs(*start*(*c_subact_2*(`nicole`)), 19)	*occurs*(*stop*(*c_subact_2*(`nicole`)), 20)
occurs(*stand_up*(`nicole`), 21)	*occurs*(*leave*(`nicole`), 22)
occurs(*stop*(*c_act*(`nicole`, `veg_r`, `lentil_soup`)), 23)	

Thus, the reader of this scenario understands that Nicole has stopped *c_subact_2* immediately after starting it because she realized that its goal is already fulfilled. Based on this answer set, questions about facts not mentioned in the narrative can be answered correctly:

Q: Did Nicole pay for the soup? A: No.
Q: Did Nicole leave the restaurant? A: Yes.

Futile Activity in Example 3. The logic form for 3 contains the following observations extracted from the text:

$st\_hpd(enter(\texttt{nicole}, \texttt{veg\_r}), \texttt{true}, 0)$.
$st\_hpd(sit(\texttt{nicole}), \texttt{true}, 1)$.
$st\_hpd(pick\_up(\texttt{nicole}, \texttt{m}, \texttt{t}), true, 2)$.
$st\_obs(available(\texttt{lentil\_soup}, \texttt{veg\_r}), false, 3)$.

The answer set for $\Pi(3)$ contains the same *occurs* predicates as $\Pi(1)$ (plus an explanation for st_obs) until step 8 and then she stops her futile activity:

$occurs(stop(c\_act(\texttt{nicole}, \texttt{veg\_r}, \texttt{lentil\_soup})), 9)$

Thus the reader is cautious and does not make any assumptions about Nicole leaving the restaurant. As expected, it does not state that Nicole ate lentil soup either, which would be impossible, nor that she paid for anything.

Diagnosis in Example 4. The logic form contains the observations:

$st\_hpd(enter(\texttt{nicole}, \texttt{veg\_r}), \texttt{true}, 0)$.
$st\_hpd(order(\texttt{nicole}, \texttt{lentil\_soup}, \texttt{waitress}), \texttt{true}, 1)$.
$st\_hpd(put\_down(\texttt{waitress}, \texttt{miso\_soup}, \texttt{t}), \texttt{true}, 2)$.

The program $\Pi(4)$ has two answer sets, containing explanations on what may have gone wrong. The first one contains the same *occurs* predicates as $\Pi(1)$ up to time step 8, and then:

$occurs(interference, 9)$
$occurs(order(\texttt{nicole}, \texttt{lentil\_soup}, \texttt{waitress}), 9)$
$occurs(move(\texttt{waitress}, \texttt{t}, \texttt{kitchen}), 10)$
$occurs(request(\texttt{waitress}, \textbf{\texttt{miso\_soup}}, \texttt{cook1}), 11)$
$occurs(prepare(\texttt{cook1}, \textbf{\texttt{miso\_soup}}, \texttt{waitress}), 12)$
$occurs(pick\_up(\texttt{waitress}, \textbf{\texttt{miso\_soup}}, \texttt{kitchen}), 13)$
$occurs(move(\texttt{waitress}, \texttt{kitchen}, \texttt{t}), 14)$
$occurs(put\_down(\texttt{waitress}, \textbf{\texttt{miso\_soup}}, \texttt{t}), 15)$
$intend(w\_seq(\texttt{waitress}, \texttt{nicole}, \textbf{\texttt{miso\_soup}}, \textbf{\texttt{miso\_soup}}, \texttt{b}), 0)$
$intend(c\_seq(\texttt{cook1}, \textbf{\texttt{miso\_soup}}, \texttt{waitress}), 0)$

Here, the reader's explanation is that there was some interference at time step 9 when Nicole ordered the lentil soup. As a result, the waitress misunderstood the order to be for miso soup and the cook followed her order.

The second answer set differs from the first in terms of what occurs at time steps 9-11 only, and the waiter and cook's intentions:

$occurs(order(\texttt{nicole}, \texttt{lentil\_soup}, \texttt{waitress}), 9)$
$occurs(move(\texttt{waitress}, \texttt{t}, \texttt{kitchen}), 10)$
$occurs(interference, 11)$
$occurs(request(\texttt{waitress}, \texttt{lentil\_soup}, \texttt{cook1}), 11)$
$intend(w\_seq(\texttt{waitress}, \texttt{nicole}, \texttt{lentil\_soup}, \textbf{\texttt{miso\_soup}}, \texttt{b}), 0)$
$intend(c\_seq(\texttt{cook1}, \textbf{\texttt{miso\_soup}}, \texttt{waitress}), 0)$

It corresponds to a second possible explanation in which the waitress understood the order correctly and the misunderstanding/ interference occurred at step 11 when she communicated the order to the cook. (Note that the reader makes no assumptions about what happens next.) The two answer sets show how reasoning by cases in ASP is useful to answering questions like:

Q: Why did the waitress bring the wrong order?
  A1: Because the waitress misunderstood the order.
  A2: Because the cook misunderstood the order.

6 Future Work and Conclusions

In this work, we proposed a new methodology of automating the understanding of narratives about stereotypical activities. The first main contribution is the use of a theory of intentions by Blount *et al.* [6] to describe the main character's intentions, which allowed reasoning correctly about exceptional scenarios, something that could not be done in previous work by Mueller [12]. Our second contribution is determining what parts of the architecture of an intentional agent, $\mathcal{AIA}$ [6], are relevant to modeling a third-person observer (a cautious reader) and what additional knowledge is required. We exemplified our methodology on several types of scenarios. In future work, we plan to view secondary characters (e.g., waiter, cook) as goal-oriented agents as well. This will require adapting Blount *et al.*'s theory to keep track of the intentions of multiple agents. We intend to create a narrative corpus similar to that of Mueller [12] and test our methodology on it. We also want to explore alternative solutions to defining futility as background knowledge, to alleviate the task of the knowledge engineer.

References

1. Balduccini, M.: CR-MODELS: An inference engine for CR-Prolog. In: Baral, C., Brewka, G., Schlipf, J.S. (eds.) Proceedings of LPNMR 2007. LNCS, vol. 4483, pp. 18–30. Springer (2007)
2. Balduccini, M., Gelfond, M.: Logic Programs with Consistency-Restoring Rules. In: Proceedings of Commonsense-03. pp. 9–18. AAAI Press (2003)
3. Balduccini, M., Gelfond, M.: The AAA architecture: An overview. In: Architectures for Intelligent Theory-Based Agents, Papers from the 2008 AAAI Spring Symposium, 2008. pp. 1–6. AAAI Press (2008)
4. Baral, C., Gelfond, M.: Reasoning about Intended Actions. In: Proceedings of AAAI-05. pp. 689–694. AAAI Press (2005)
5. Blount, J.: An Architecture for Intentional Agents. Ph.D. thesis, Texas Tech University, Lubbock, TX, USA (2013)
6. Blount, J., Gelfond, M., Balduccini, M.: A theory of intentions for intelligent agents. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Proceedings of LPNMR 2015. LNCS, vol. 9345, pp. 134–142. Springer (2015)
7. Gabaldon, A.: Activity recognition with intended actions. In: Boutilier, C. (ed.) Proceedings of IJCAI 2009. pp. 1696–1701 (2009)
8. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents. Cambridge University Press (2014)
9. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3/4), 365–386 (1991)
10. Inclezan, D., Gelfond, M.: Representing Biological Processes in Modular Action Language ALM. In: Proceedings of Commonsense 2011. pp. 49–55 (2011)
11. Mueller, E.T.: Understanding script-based stories using commonsense reasoning. Cognitive Systems Research 5(4), 307–340 (2004)
12. Mueller, E.T.: Modelling space and time in narratives about restaurants. Literary and Linguistic Computing 22(1), 67–84 (2007)
13. Schank, R.C., Abelson, R.P.: Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures. Lawrence Erlbaum (1977)
14. Shanahan, M.: Solving the Frame Problem. MIT Press (1997)

A Partial Output for the Normal Scenario in Example 1

$map(0, 2)$
$map(1, 9)$
$map(2, 15)$
$map(3, 17)$
$map(4, 27)$
$occurs(select(satiated\_and\_out(\texttt{nicole})), 0)$
$occurs(start(c\_act(\texttt{nicole}, \texttt{veg\_r}, \texttt{lentil\_soup})), 1)$
$occurs(enter(\texttt{nicole}, \texttt{veg\_r}), 2)$
$occurs(greet(\texttt{waitress}, \texttt{nicole}), 3)$
$occurs(lead\_to(\texttt{waitress}, \texttt{nicole}, \texttt{t}), 4)$
$occurs(sit(\texttt{nicole}), 5)$
$occurs(start(c\_subact\_1(\texttt{nicole}, \texttt{lentil\_soup})), 6)$
$occurs(pick\_up(\texttt{nicole}, \texttt{m}, \texttt{t}), 7)$
$occurs(put\_down(\texttt{nicole}, \texttt{m}, \texttt{t}), 8)$
$occurs(order(\texttt{nicole}, \texttt{lentil\_soup}, \texttt{waitress}), 9)$
$occurs(move(\texttt{waitress}, \texttt{t}, \texttt{kitchen}), 10)$
$occurs(request(\texttt{waitress}, \texttt{lentil\_soup}, \texttt{cook1}), 11)$
$occurs(prepare(cook1, \texttt{lentil\_soup}, \texttt{waitress}), 12)$
$occurs(pick\_up(\texttt{waitress}, \texttt{lentil\_soup}, \texttt{kitchen}), 13)$
$occurs(move(\texttt{waitress}, \texttt{kitchen}, \texttt{t}), 14)$
$occurs(put\_down(\texttt{waitress}, \texttt{lentil\_soup}, \texttt{t}), 15)$
$occurs(stop(c\_subact\_1(\texttt{nicole}, \texttt{lentil\_soup})), 16)$
$occurs(eat(\texttt{nicole}, \texttt{lentil\_soup}), 17)$
$occurs(start(c\_subact\_2(\texttt{nicole})), 18)$
$occurs(request(\texttt{nicole}, \texttt{b}, \texttt{waitress}), 19)$
$occurs(move(\texttt{waitress}, \texttt{t}, \texttt{counter}), 20)$
$occurs(pick\_up(\texttt{waitress}, \texttt{b}, \texttt{counter}), 21)$
$occurs(move(\texttt{waitress}, \texttt{counter}, \texttt{t}), 22)$
$occurs(put\_down(\texttt{waitress}, \texttt{b}, \texttt{t}), 23)$
$occurs(pay(\texttt{nicole}, \texttt{b}), 24)$
$occurs(stop(c\_subact\_2(\texttt{nicole})), 25)$
$occurs(stand\_up(\texttt{nicole}), 26)$
$occurs(leave(\texttt{nicole}), 27)$
$occurs(stop(c\_act(\texttt{nicole}, \texttt{veg\_r}, \texttt{lentil\_soup})), 28)$