AAAI Technical Report WS-11-18

A Discrete Event Calculus Implementation of the OCC Theory of Emotion

Margaret Sarlej and **Malcolm Ryan**
Department of Computer Science and Engineering
University of New South Wales
Sydney, NSW, Australia
msarlej@cse.unsw.edu.au, malcolmr@cse.unsw.edu.au

Abstract

Characters are a critical part of storytelling and emotion is a vital part of character. Readers generally credit characters with human emotions, and it is these emotions which bring meaning to stories. To computationally construct interesting and meaningful stories we need a model of emotion which allows us to predict characters' reactions to events in the world. There are many different psychological theories of emotion; the most popular to date for computational applications is the OCC theory. This paper describes a Discrete Event Calculus implementation of the OCC Theory of Emotion. To evaluate our system, we apply it to a selection of Aesop's fables, and compare the output to the emotions readers expect in the same situations based on a survey.

Introduction

The most basic elements of storytelling are setting, plot, and character. All three provide significant challenges for computational story understanding and generation even in the simplest of stories (Ryan, Hannah, and Lobb 2007). In many stories, these elements combine to convey a moral or message. Morals were central to why storytelling first developed, as a method of communication (Ryan 1991). We believe morals can be represented using the patterns of emotion experienced by characters in a story. Shaheed and Cunningham (2008) also identify a relationship between emotions and morals, but with a focus on designing agents. Our eventual goal is to develop a system able to generate stories that express a specified moral; the first step to this end is building a framework to model emotion.

Realistically modelling emotion involves a deep understanding of human psychology. There have been many theories of emotion (or *affect*) proposed over the years (Izard 1977; Plutchik 1980; Russell 1980; Frijda 1986; Lazarus 1991), but few are amenable to computational modelling due to a general informality of the psychological descriptions. One important exception is the OCC Theory of Emotion (Ortony, Clore, and Collins 1988), which was specifically designed to be computationally modelled, and hence provides a suitable foundation for our work.

Most of the previous work based on the OCC theory has focussed on developing emotional agents (Elliott 1992; Reilly 1996; Gratch and Marsella 2004). In the context of narrative, this translates to characters that react appropriately to circumstances based on emotions they experience; i.e. a character-centric approach. We take a plot-centric approach; our system models characters' emotions as events occur, but does not use them for decision-making on the part of the characters. Rather, the aim is to specify a pattern of emotions and have the system generate sequences of events which would cause the characters to experience those emotions. This is similar to how a human author might manipulate characters' emotions through particular plot choices.

OCC Theory of Emotion

The OCC theory (Ortony, Clore, and Collins 1988) divides emotions into three main categories: event-based emotions, agent-based emotions and object-based emotions. There is also a group of compound emotions.

Event-based emotions Event-based emotions are experienced as a result of the consequences of events, either for the self or for another agent. Those directed towards the self include Joy, Distress, Satisfaction, Fears-Confirmed, Relief and Disappointment. Those directed towards others include Happy-For, Pity, Gloating and Resentment.

Agent-based emotions Agent-based emotions are felt towards an agent (possibly the self) with respect to some action. Pride and Shame are directed towards the self, Admiration and Reproach towards other agents.

Object-based emotions Object-based emotions are those directed towards objects (agents or inanimate objects). The two emotions in this category are Love and Hate.

Compound emotions While most emotions fit into one of the above categories, there are four compound emotions which are simultaneously event- and agent-based. These emotions are felt towards an agent with respect to the consequences of an event. Gratification and Remorse are directed towards the self; Gratitude and Anger are directed towards other agents.

Adam, Herzig and Longin (2009) provide a logical formalisation of the OCC theory using modal logic. They handle 20 of the 22 emotions described by the OCC theory, excluding only the object-based emotions of Love and Hate. We use some of their ideas in our implementation, in particular with regards to representing event-based emotions.

The Event Calculus

We base our implementation on the Discrete form of the Event Calculus (Mueller 2006). The Event Calculus was developed to handle problems involving commonsense reasoning about events and their effects. People rely on this in their daily lives, and naturally expect what happens in a story to be consistent with common sense. If we specify a range of conditions on a story (e.g. which emotions must be generated), commonsense reasoning can be used to fill in the missing information (i.e. which events could have taken place).

The Event Calculus is based on events and fluents. An *event* is something that happens at a particular point in time. Events can affect the values of fluents. A *fluent* is a property of the world, which can be true or false at any particular timepoint. To resolve the frame problem, fluents follow the *commonsense law of inertia*; i.e. their truth value remains the same unless affected by an event. Mueller describes two forms of the Event Calculus: Continuous and Discrete. The difference is in their treatment of time; the Discrete Event Calculus only allows integer time values, treating time as a sequence of discrete steps. For storytelling, we see no need to handle continuous time. Our only requirement is to be able to identify the order in which events happen, which can be achieved just as effectively using discrete time-steps.

The Discrete Event Calculus Reasoner

Mueller provides an implementation of the Discrete Event Calculus, the *Discrete Event Calculus Reasoner*, or *decreasoner* (Mueller 2008), which we have used to construct our system. It provides a syntax based on the Event Calculus which can be used to specify various rules and conditions to form a domain description. This domain description is translated by decreasoner into a satisfiability (SAT) problem, which is then run through a standard SAT solver.

The Discrete Event Calculus Reasoner can perform both *deduction* (determining the resulting situation based on an initial situation and a sequence of events) and *abduction* (determining a sequence of events based on an initial situation and a final situation). The output of the program is a set of models which are consistent with all axioms and facts provided in the domain description. Ultimately we intend to use decreasoner's abductive capabilities to generate sequences of events that comply with a range of conditions on character emotions. However, in this study we only aim to evaluate our preliminary emotion model, and thus use deduction to generate character emotions resulting from sequences of events based on Aesop's fables (Aesop 1998).

Modelling Emotions using the Discrete Event Calculus Reasoner

Our implementation is a simplified version of the OCC theory. We exclude the prospect-based emotions Satisfaction, Fears-Confirmed, Relief and Disappointment. This removes the need to consider characters' expectations, thereby reducing the information we need to model. We recognise this is a significant limitation, and intend to extend the system to cater for this category of emotions. We have also chosen not to model the *intensity variables* identified by Ortony, Clore and Collins (1988). To keep the system simple (as appropriate to dealing with fables, which are some of the simplest of stories) we deal with absolutes rather than degrees of emotion. We do believe, however, that incorporating these variables will be required to model stories in which the interactions between characters' emotions are more complex.

Assumptions

1. Similarly to Adam, Herzig and Longin (2009) we assume actions are carried out with intent. This allows agents to attribute blame for things that happen, facilitating emotions such as Gratitude and Anger. We do not model accidents or events with no instigator, though this will undoubtedly be necessary for any realistic application.
2. We treat emotions as momentary rather than persistent. Consider an agent a who experiences Joy as a result of some event e_1, but is not emotionally affected by some other event e_2. If e_1 occurs at time=0, agent a would feel Joy at time=1. If event e_2 then takes place at time=1, a will not feel any emotion at time=2, because the Joy from e_1 will not persist past time=1. This is in line with Adam, Herzig and Longin's assertion that "an affective state having a long duration is not so much emotion as mood." The exception is the object-based emotions, which persist until an event changes them (see *Like and Dislike*).
3. We assume all agents in the location where an event occurs have perfect knowledge of the event and, more importantly, share the same emotional interpretation of the event. That is, all agents agree on which emotions a particular agent would feel in response to a particular event.
4. We permit only one action to occur at any given timepoint. This avoids the need for encoding restrictions about whether agents can be involved in two actions simultaneously, and is also in line with the way simple stories are structured, i.e. events occur in sequence.

Events and their Consequences

The most obvious approach to representing events would be to define each as an instance of decreasoner's *event* sort. The disadvantage is that this would require the emotional effects of each event to be defined separately. This is unnecessary duplication; many different events can have the same emotional effects on the agents involved. Instead, we define an *eventname* sort. This is passed as a parameter to *DoActionSingle* and *DoActionDouble*, which are of the event sort and apply to single-agent and double-agent actions respectively:

```
DoActionSingle(eventname,agent)
DoActionDouble(eventname,agent1,agent2)
```

To group events according to the emotions they incite, we define *emotion classes*. For single-agent events (i.e. events that involve and/or affect only a single agent), there are only three possible classes, summarised in Table 1. For events involving (or affecting) two agents, there are nine such classes, summarised in Table 2. This could be extended to three agents (27 emotion classes) or more, with the number of emotion classes determined by the permutations of the three possibilities Joy, Distress and None.

Figure 1 Initiation conditions for Joy and Distress based on an event of emotion class JD

```
[agent1,agent2,time,consequence]
    Initiates(JD(agent1,agent2,consequence),Joy(agent1,consequence),time).
[agent1,agent2,time,consequence]
    Initiates(JD(agent1,agent2,consequence),Distress(agent2,consequence),time).
```

Emotion Class	Effect on Agent
J	Joy
D	Distress
N	None

Table 1: Emotion classes for single-agent events

Emotion Class	Agent 1	Agent 2
JJ	Joy	Joy
JD	Joy	Distress
DJ	Distress	Joy
DD	Distress	Distress
JN	Joy	None
NJ	None	Joy
DN	Distress	None
ND	None	Distress
NN	None	None

Table 2: Emotion classes for double-agent events

We limit our current implementation to at most two agents involved in a single event. This is sufficient for dealing with fables, as individual events generally do not involve more than two characters directly. Some fables refer to groups of characters, for example *the villagers* in Aesop's Fable 318 (Aesop 1998); in these instances, the group can be defined as a single agent because the emotional effects on all members of the group are the same.

However, simply allocating events into one of these classes would limit each event to having a single emotional effect on any particular agent. Adam, Herzig and Longin (2009) identify that one of the difficulties in formalising the OCC theory logically is that a single event can result in both positive and negative emotions. They suggest focussing on the "desirability of *consequences of events*" rather than the events themselves. We have adopted this approach for handling event-based emotions, and to this end define a *consequence* sort. Each event can be associated with one or more consequences, and it is the consequence rather than the eventname which is allocated to one of the emotion classes presented in Tables 1 and 2. Events are mapped to one or more consequences using the *IsConsequenceOf* fluent:

```
IsConsequenceOf(consequence,eventname)
```

Each consequence then needs to be categorised into the appropriate emotion class, which is achieved using fluents such as:

```
IsJ(consequence)
    ...
IsJJ(consequence)
IsJD(consequence)
    ...
```

The Basic Emotions

We will refer to those emotions which do not depend on other emotions as *basic emotions*. They are modelled in our system as inertial fluents, and therefore require explicit initiation and termination conditions. We consider Joy, Distress, Admiration, Reproach, Like and Dislike to be basic emotions.

Joy and Distress Joy and Distress are the simplest emotions, determined solely by the emotion classes of the consequences of events that take place. *Initiates* and *Terminates* predicates are defined to handle these emotions for each emotion class. Because the same event can have both positive and negative consequences, Joy and Distress must be parameterised in terms of the consequence that caused the emotion:

```
     Joy(agent,consequence)
Distress(agent,consequence)
```

Figure 1 shows the initiation conditions for Joy and Distress resulting from a JD consequence.

Admiration and Reproach Admiration and Reproach are also treated inertially, based on a simple principle. If an agent carries out an action which benefits another agent (i.e. causes them Joy), this action is admired by other agents; if an agent causes another agent Distress, this incites reproach from other agents. Because these emotions are directed towards an event (not its consequences), they do not require a consequence parameter:

```
Admiration(agent1,agent2)
  Reproach(agent1,agent2)
```

The difficulty is that Joy and Distress, which determine Admiration and Reproach, result from consequences of events. This is particularly problematic when a single event has both positive and negative consequences. Our solution is to define the following predicates:

```
HasAdmirationPotential(eventname)
  HasReproachPotential(eventname)
```

In simple terms, if an action has a consequence which causes Joy for another agent, the action has *admiration potential*; if an action has a consequence which causes Distress for another agent, it has *reproach potential*. On this basis we define non-inertial fluents *IsAdmirable* and *IsReproachable*. IsAdmirable holds as long as an action has admiration potential but not reproach potential, and vice versa for IsReproachable. These predicates are then used as conditions to initiate Admiration and Reproach respectively. Figure 2 shows the decreasoner rules corresponding to Admiration.

Figure 2 Definition of IsAdmirable and initiation condition for Admiration

```
[time,eventname]
    HoldsAt(HasAdmirationPotential(eventname),time) &
   !HoldsAt(HasReproachPotential(eventname),time) <-> HoldsAt(IsAdmirable(eventname),time).
[agent1,agent2,agent3,time,eventname]
    HoldsAt(Colocated(agent1,agent3),time) & agent1 != agent3 &
    HoldsAt(IsAdmirable(eventname),time)
        -> Initiates(DoActionDouble(eventname,agent1,agent2),Admiration(agent3,agent1),time).
```

Figure 3 Like and Dislike

```
[agent1,agent2,time,eventname] HoldsAt(IsAdmirable(eventname),time)
    -> Initiates(DoActionDouble(eventname,agent1,agent2),Like(agent2,agent1),time).
[agent1,agent2,time,eventname] HoldsAt(IsAdmirable(eventname),time)
    -> Terminates(DoActionDouble(eventname,agent1,agent2),Dislike(agent2,agent1),time).
```

This approach is, of course, limited. A complete implementation should provide a more flexible notion of 'standards,' similar to the social 'norms' described by Damiano (2002). Standards should also be able to vary between agents, to reflect the way that in real life the same action can be considered admirable by one person but reproachable by another, depending on their individual concept of morality.

Like and Dislike Like and Dislike are the predicates we define as equivalent to the OCC object-based emotions Love and Hate:

```
   Like(agent,object)
Dislike(agent,object)
```

Objects can, and in our system invariably do, include other agents, because we deal with the emotional layer only. We treat object-based emotions differently from the other categories in that we allow them to persist until an event explicitly changes them. This is because of the role they play in determining other emotions. For example, consider agents a_1 and a_2. If a_1 experiences Distress due to some event, one would expect a_2 to feel Pity for a_1, *except* if a_2 dislikes a_1, in which case they would experience Gloating.

In the first version of our system, we implemented Like and Dislike as fixed based on their value at time=0. This is analogous to a strong friendship or enmity, which won't be swayed by basic events. We later modified this to allow Like and Dislike to vary based on events that take place using a simple definition; an admirable event initiates Like while a reproachable event initiates Dislike. Figure 3 shows the condition for initiating Like (and hence terminating Dislike).

Agent Beliefs

To effectively represent those emotions which depend on other characters' emotions we also need to model beliefs. To see why, consider the fortunes-of-others emotion *Happy-For*. For an agent a_1 to feel Happy-For some other agent a_2, a_1 must believe a_2 is feeling Joy; otherwise the Happy-For emotion doesn't make sense.

A way around this without modelling beliefs explicitly would be to assume all agents have perfect knowledge of other agents' emotions. This would allow a_1 to feel Happy-For a_2 based on a_2 feeling Joy, without any need for a_1 to have a belief about a_2's Joy. However, this is less flexible and also less realistic; in reality, people often make assumptions about what others are feeling, but these assumptions are not always correct. Incorrect assumptions about others' emotions can lead to interesting story patterns, and thus we have built the mechanisms to handle this into our model.

For each emotion, we define a secondary fluent which represents other agents' beliefs. For example, the normal fluent which models Joy is:

```
Joy(agent,consequence)
```

The corresponding belief fluent is:

```
BJoy(agent1,agent2,consequence)
```

In our current system, the belief fluents mirror the values of the normal emotion fluents because of our assumption that agents share the same emotional interpretation of events (refer to *Assumptions*). To keep computational complexity to a minimum, we limit beliefs to one level. An agent can have beliefs about another agent's emotions, but cannot have beliefs about other agents' beliefs. A limitation of this approach is that we cannot represent beliefs about some of the complex emotions, because that would require a second meta-level of belief.

Complex Emotions

Complex emotions are those that depend on other emotions. We model them as non-inertial fluents, which allows them to vary freely according to the values of those emotions they depend on. There are ten complex emotions in our model: Happy-For, Pity, Gloating, Resentment, Pride, Shame, Gratification, Remorse, Gratitude and Anger.

Happy-For, Pity, Gloating, Resentment The fortunes-of-others emotions are directed towards other agents, with respect to some consequence:

```
  HappyFor(agent1,agent2,consequence)
      Pity(agent1,agent2,consequence)
  Gloating(agent1,agent2,consequence)
Resentment(agent1,agent2,consequence)
```

Each of these emotions depends on two factors: a Like or Dislike value, and a belief about another agent's Joy or Distress. In our implementation an agent must like another

Figure 4 The fortunes-of-others emotions: Happy-For, Pity, Gloating and Resentment

```
[agent1,agent2,time,consequence]
  HoldsAt(Like(agent1,agent2),time) & HoldsAt(BJoy(agent1,agent2,consequence),time)
      <-> HoldsAt(HappyFor(agent1,agent2,consequence),time).
[agent1,agent2,time,consequence]
  !HoldsAt(Dislike(agent1,agent2),time) & HoldsAt(BDistress(agent1,agent2,consequence),time)
      <-> HoldsAt(Pity(agent1,agent2,consequence),time).
[agent1,agent2,time,consequence]
  HoldsAt(Dislike(agent1,agent2),time) & HoldsAt(BDistress(agent1,agent2,consequence),time)
      <-> HoldsAt(Gloating(agent1,agent2,consequence),time).
[agent1,agent2,time,consequence]
  HoldsAt(Dislike(agent1,agent2),time) & HoldsAt(BJoy(agent1,agent2,consequence),time)
      <-> HoldsAt(Resentment(agent1,agent2,consequence),time).
```

Figure 5 Pride

```
[agent1,agent2,time]
    HoldsAt(BAdmiration(agent1,agent2,agent1),time) &  !HoldsAt(Dislike(agent1,agent2),time)
        -> HoldsAt(Pride(agent1),time).
[agent1,time] {agent2} HoldsAt(Pride(agent1),time) ->
    HoldsAt(BAdmiration(agent1,agent2,agent1),time) & !HoldsAt(Dislike(agent1,agent2),time).
```

Figure 6 Gratification and Remorse

```
[agent,time,consequence] HoldsAt(Pride(agent),time) &
    HoldsAt(Joy(agent,consequence),time) <-> HoldsAt(Gratification(agent,consequence),time).
[agent,time,consequence] HoldsAt(Shame(agent),time) &
    HoldsAt(Distress(agent,consequence),time) <-> HoldsAt(Remorse(agent,consequence),time).
```

agent to feel Happy-For them; however, the only condition for Pity is that they don't dislike the other agent. Both Gloating and Resentment require that an agent dislike the other agent; the only difference is whether they believe the other agent to be experiencing Joy or Distress. Figure 4 shows the conditions for each of these emotions.

Pride and Shame Pride and Shame are self-directed emotions in response to an event, not its consequences. Because we only allow one event to take place at any given timepoint, we only need to parameterise these emotion predicates with the agent involved; the event to which it relates can be inferred as that which took place in the most recent timepoint.

```
Pride(agent)
Shame(agent)
```

Britt and Heise (2000) summarise Scheff's (1990) definitions of pride and shame as follows:

> ... shame occurs when one feels negatively evaluated by self or others, while pride is evident when one feels positively evaluated by self or others.

This is in line with Stage 3 of Kohlberg's theory of moral development (Kohlberg 1958), where individuals determine right and wrong based on the approval or disapproval of others. In our model, an agent feels Pride if they believe another agent feels Admiration towards them (provided they do not dislike this other agent). Similarly, an agent feels Shame if they believe another agent (whom they don't dislike) feels Reproach towards them. The conditions for Pride are shown in Figure 5; Shame is identical with respect to BReproach.

Gratification and Remorse Pride and Shame, in turn, are used to trigger the compound emotions Gratification and Remorse:

```
Gratification(agent,consequence)
      Remorse(agent,consequence)
```

If an agent feels Pride with respect to their own action, and experiences Joy as a result of one of that action's consequences, they feel Gratification. Conversely, if an agent feels Shame about one of their own actions, and experiences Distress from one of its consequences, the agent feels Remorse. This is shown in Figure 6.

Gratitude and Anger Compound emotions such as Gratitude and Anger are directed towards another agent's action with respect to its outcomes:

```
Gratitude(agent1,agent2,consequence)
    Anger(agent1,agent2,consequence)
```

If agent a_1 feels Admiration about agent a_2's action and experiences Joy from one of its consequences, a_1 feels Gratitude towards a_2. On the other hand, if a_1 feels Reproach about a_2's action and experiences Distress from one of its consequences, a_1 feels Anger towards a_2. The rules for these emotions are shown in Figure 7.

Location

Our current model reasons only about the emotional level of events, and therefore does not consider physical constraints on actions. The only exception to this is location. For an event involving two agents to take place, both agents must be

Figure 7 Gratitude and Anger

```
[agent1,agent2,time,consequence]
    HoldsAt(Admiration(agent1,agent2),time) & HoldsAt(Joy(agent1,consequence),time)
        <-> HoldsAt(Gratitude(agent1,agent2,consequence),time).
[agent1,agent2,time,consequence]
    HoldsAt(Reproach(agent1,agent2),time) & HoldsAt(Distress(agent1,consequence),time)
        <-> HoldsAt(Anger(agent1,agent2,consequence),time).
```

in the same location. Similarly, for an agent to feel emotions towards another agent following an event (whether involved in the event or not), they must be aware the event took place. Our assumption that all agents in the location where an event happens are aware of the event requires us to model each agent's location:

```
Located(agent,location)
```

To simplify the conditions we also define a non-inertial fluent, *Colocated*, which holds if Located holds for two different agents with the same location:

```
Colocated(agent1,agent2)
```

Initial State Information and Conditions

In order for the program to work, certain information needs to be provided in the domain description:

- The list of available agents, locations, eventnames and consequences.
- A mapping of consequences to eventnames, using *IsConsequenceOf*.
- A mapping of consequences to their emotion classes, using *IsJ*, *IsD*, *IsJD*, etc.

Information about which fluents hold in the initial state can also optionally be specified. For example, if no agent is to be feeling Joy at time=0, this can be specified as follows:

```
[agent,consequence]
    !HoldsAt(Joy(agent,consequence),0).
```

More general conditions that hold at all timepoints can also be included. For example, if Jack dislikes Jill at all times during a story, this can be specified using:

```
[time]
    HoldsAt(Dislike(Jack,Jill),time).
```

In decreasoner syntax square brackets [ ] denote the universal quantifier $\forall$. So, in the example above, *Dislike(Jack,Jill)* holds for all possible values of *time*. The existential quantifier $\exists$ is represented using braces { }. If Jack must like Jill at some time during a story, this would be represented by:

```
{time}
    HoldsAt(Like(Jack,Jill),time).
```

If the goal is to determine which emotions characters should experience during a predetermined story, or if the story must include one or more specific events, the event to take place at any timepoint can be specified using the *Happens* predicate. For example:

```
Happens(DoActionDouble(Torments,Wasp,
                                Snake),0).
```

Evaluation

To evaluate the effectiveness of our model compared to the emotions people would expect characters to feel in the same situations, we conducted a short survey based on Aesop's fables (Aesop 1998). Participants were provided with six fables and asked to select which of the OCC emotions they expected characters to feel after each event. Because emotion words in any language can be ambiguous, participants were provided with definitions in their list of options.

A total of 25 participants took part in the survey. All participants responded to the first fable, 23 responded to the second and 17 completed the entire survey. We recognise that such a small sample size won't yield statistically significant results. Nevertheless we believe the responses, particularly those with a high level of agreement between participants, will provide a useful indication of the emotions readers would expect characters to feel.

We compared the survey results to three versions of our system:

1. Version 1 treats Like and Dislike as fixed at time=0, and uses our own assignment of consequences to emotion classes.
2. Version 2 also treats Like and Dislike as fixed, but this time consequences are assigned to emotion classes based on participant responses. For example, if the majority of participants believed that, as a result of a Wasp tormenting a Snake, the Wasp would feel Joy and the Snake would feel Distress, we classified the action as JD. This allows us to see whether the other emotions are entailed from Joy and Distress in line with readers' expectations.
3. Version 3 allows Like and Dislike to vary based on admirable and reproachable actions. This is significant because Like and Dislike play a role in determining the fortunes-of-others emotions. Emotion classes are assigned to match participant responses, as in Version 2.

Analysis

To evaluate our system we require a baseline for comparison; we define this based on our survey results. We decompose each question into the available emotion options such that each can be treated as a binary classification; i.e. true if the emotion was generated in that circumstance, false if not. In each case, the majority response in the survey is considered to be the prediction of an ideal classifier, represented as a binary vector. This is the best possible result given the noise in the data due to participants' differing opinions.

In comparing the performance of our system to the baseline classifier we consider two types of error:

Version	Error Rates (False Positive / False Negative)			
	Overall	Joy/Distress	Like/Dislike	Other Emotions
Baseline	4.6% / 4.2%	5.2% / 3.0%	4.5% / 7.0%	4.7% / 4.0%
Version 1	7.4% / 18.4%	1.7% / 12.3%	10.8% / 31.3%	7.9% / 17.3%
Version 2	10.7% / 16.2%	6.4% / 5.1%	10.8% / 31.3%	11.5% / 16.1%
Version 3	10.6% / 12.0%	6.4% / 5.1%	16.5% / 12.8%	11.1% / 14.2%

Table 3: Error rate comparison to baseline classifier

1. **False Positives:** Instances in which the system predicts an emotion which was not reported by a user.
2. **False Negatives:** Instances where the system fails to predict an emotion predicted by a user.

To calculate the error rates for the ideal classifier, we compare its binary vector to the vector obtained from each participant's survey response, summing the false positives and false negatives across all events in all fables. This figure is reported as a percentage of the total number of emotions possible across all six fables. Error rates for the three versions of our system were calculated in the same manner.

In addition to an overall error rating, we also provide the error rates for three distinct subgroups of emotions:

1. **Joy/Distress:** Only Joy and Distress are considered. The error rates were expected to match between Versions 2 and 3, because the emotion class assignments were identical in both versions
2. **Like/Dislike:** Only Like and Dislike are considered. The error rates were expected to match between Versions 1 and 2, because Like and Dislike were fixed to the same values in both versions.
3. **Other Emotions:** All emotions except Joy, Distress, Like and Dislike. This isolates how changes to the behaviour of those four emotions affect other more complex emotions.

Results

Table 3 summarises our results. Overall, Version 3 is the most successful; all but one of its error rates are within 10% of the minimum error rates set by the baseline classifier. Comparing Versions 1 and 2, generally the false negative rate decreases, but the false positive rate increases. The exception is Like and Dislike, because they are fixed (to the same values) in both versions. Between Versions 2 and 3 this is reversed; Like and Dislike vary, while Joy and Distress remain the same, because the emotion class assignments are identical between these versions.

The most significant drop in false negatives is between Versions 2 and 3. Part of this improvement can be attributed to the correct Like and Dislike values being generated (note the drop in false negatives between Versions 2 and 3 for Like/Dislike). However, there is also an improvement for the other emotions, both in reducing false negatives and, to a lesser extent, false positives as well. This highlights the importance of Like and Dislike in determining complex emotions; it is likely overall performance could be further improved by correcting their behaviour.

The other point worth noting is that the most significant increase in false positives appears between Versions 1 and 2. The only difference between them is the Joy and Distress classifications assigned to the consequences of events; all the rules and axioms are the same. This seems to indicate that correctly classifying consequences is critical for producing the desired emotional effects. However, our survey results show that many participants disagreed on what the correct classifications should be, making it impossible for a system to cater for all interpretations simultaneously. This is a good illustration of the difficulties that arise in a field as subjective and ambiguous as emotion; hardly surprising, given that ambiguity is a poor companion for logic.

We also made a number of observations based on our comparison of the decreasoner output with the survey responses, which will need to be considered in future developments of the system:

- The same consequence can be categorised differently at different times in the same story. For example, survey participants classed losing offspring as DD when the Eagle eats the Fox's cubs at the beginning of Fable 3 (Aesop 1998); they are friends, so this would cause Distress for both. However, the same consequence is classed as JD when the Fox later eats the eaglets. Readers assume the relationship between the pair has changed by this point, and thus the Fox should feel Joy at the Eagle's Distress. This explains why the Joy/Distress error rates for Versions 2 and 3 don't match the baseline classifier, despite emotion classes being matched to survey responses.
- In determining when Like and Dislike should change, there is an important distinction between an action performed with its consequence intended and an action where the consequence is not intended. In Fable 3, when the Eagle's action causes the eaglets to fall from the tree, this is a positive outcome for the Fox. However, it should not make the Fox like the Eagle, since the Eagle didn't intend that consequence when she carried out the action.
- When both positive and negative consequences are experienced, readers still expect characters to feel Anger, a prerequisite of which is Reproach. The system currently generates neither Reproach nor Admiration when consequences of the same action result in conflicting emotions.
- Making Pride and Shame dependant on the beliefs of others makes it impossible for a character to feel Pride or Shame when they are alone. Alternative definitions need to be provided to cater for single-agent actions.
- People seem to consider characters as generally selfish.

For example, participants did not expect characters to feel Happy-For or Pity towards others unless they explicitly liked them. Our definitions were less stringent, i.e. that characters would feel these emotions provided they didn't dislike the other party.

- Participant responses are not always consistent with the emotion definitions provided. For example, only 50% of respondents believed the Eagle would feel Distress about the Fox no longer being hungry in Fable 3; the rest believed the Eagle should experience no emotions about this consequence. However, 88% of respondents indicated the Eagle should feel Anger about the Fox no longer being hungry, even though according to the definition provided it is a prerequisite for Anger that the Eagle feel Distress about the outcome. This highlights the difficulties caused by the ambiguity inherent in emotion words, even when definitions are provided.

Future Work

The observations above identify several areas for improvement. It is evident that we need a more flexible model for consequences, to allow them to have different emotional effects depending on the context in the story when they occur, and also to differentiate between intentional and accidental consequences. We also need to reconsider and perhaps broaden our definitions of Anger and Gratitude as well as Pride and Shame. In addition there are the obvious extensions of including the prospect-based emotions (Satisfaction, Fears-Confirmed, Relief and Disappointment) and incorporating the OCC intensity variables. Finally, we hope to build in a way to represent individual concepts of morality, so that characters can have varying moral standards. This will allow us to model the difference between 'good' characters and 'bad' characters, which is vital to storytelling even in its simplest form.

After making these improvements, the next stage will be to develop representations for common morals in terms of patterns of emotions. Once established, these can be added to the domain description of our decreasoner implementation, allowing us to generate stories with specified morals using abduction. This will bring storytelling systems one step closer to conveying morals through narrative.

Conclusion

In this paper we presented a Discrete Event Calculus implementation of a subset of the OCC Theory of Emotion. Our survey-based evaluation identifies a number of limitations of our current system, but also shows it can achieve good results compared to a baseline error measurement computed from the survey data.

We believe, with some key modifications to resolve the issues identified above, our system could provide a reasonable approximation of the emotions readers would expect characters to feel in response to events in stories. The biggest difficulty will be in generating those emotions about which even people disagree, but that is to be expected in dealing with a field as subjective as emotion. We don't expect that perfect accuracy will be required to generate stories which effectively convey morals.

References

Aesop. 1998. *Aesop: The Complete Fables*. Penguin Classics. London, UK: Penguin Books.

Britt, L., and Heise, D. 2000. From shame to pride in identity politics. In Stryker, S.; Owens, T. J.; and White, R. W., eds., *Self, Identity, and Social Movements*. Minneapolis: University of Minnesota Press. 252–268.

Damiano, R. 2002. *The Role of Norms in Intelligent Reactive Agents*. Ph.D. Dissertation, Università degli Studi di Torino.

Elliott, C. 1992. *The Affective Reasoner: A process model of emotions in a multi-agent system*. Ph.D. Dissertation, Northwestern University.

Frijda, N. H. 1986. *The Emotions*. New York: Cambridge University Press.

Gratch, J., and Marsella, S. 2004. A Domain-independent Framework for Modeling Emotion. *Journal of Cognitive Systems Research* 5(4):269–306.

Izard, C. E. 1977. *Human Emotions*. New York: Plenum Press.

Kohlberg, L. 1958. *The Development of Modes of Thinking and Choices in years 10 to 16*. Ph.D. Dissertation, University of Chicago.

Lazarus, R. S. 1991. *Emotion and Adaptation*. New York: Oxford University Press.

Mueller, E. T. 2006. *Commonsense Reasoning*. San Francisco, CA: Morgan Kaufmann Publishers.

Mueller, E. T. 2008. *Discrete Event Calculus Reasoner Documentation*. IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Ortony, A.; Clore, G. L.; and Collins, A. 1988. *The Cognitive Structure of Emotions*. Cambridge: Cambridge University Press.

Plutchik, R. 1980. *Emotion: A Phsychoevolutionary Synthesis*. New York: Harper & Row.

Reilly, W. S. N. 1996. *Believable Social and Emotional Agents*. Ph.D. Dissertation, Carnegie Mellon University.

Russell, J. A. 1980. A circumplex model of affect. *Journal of Personality and Social Psychology* 39(6):1161–1178.

Ryan, M.; Hannah, N.; and Lobb, J. 2007. The tale of peter rabbit: a case-study in story-sense reasoning. In *Proceedings of the 4th Australasian Conference on Interactive Entertainment*. Melbourne, Australia: RMIT University.

Ryan, K. 1991. The narrative and the moral. *The Clearing House* 64(5):316–319.

Scheff, T. J. 1990. Socialization of emotions: pride and shame as causal agents. In Kemper, T. D., ed., *Research Agendas in the Sociology of Emotions*. Albany: State University of New York Press.

Shaheed, J., and Cunningham, J. 2008. Agents making moral decisions. In *Proceedings of the ECAI08 Workshop on Artificial Intelligence in Games*.