(1) also gives author assurance of coverage by ~~allowing~~ avoiding procedural description in favour of declarative

(2) Need more outline/justification for content (of rest of chapter)

Chapter 4

Logic, Norms and Institutions as Story Worlds

1.1 Describing Stories With Logic

Section ?? describes approaches that use formal logic to model interactive, non-linear narratives. Building on the work described in that section, we explore other possible ways to construct these kinds of stories.

Our end goal is to make tools which assist in the creation of non-linear, interactive narratives for games by generating paths through the stories. This saves the author the time and effort required to author all of the individual paths through the stories themselves. When generating these story branches, we need some way to check whether or not each branch "fits" with the narrative. In a story with a hero and a villain, it would not seem right if the hero were to kill innocent civilians while the villain tries to rescue them, for example. Our approach ~~to this~~ is to model the story world using formal logic, which allows us to check whether or not actions that occur within the story are consistent with its description.

Using the *Punch and Judy* story world as our example, we formalise its narrative rules by describing them in terms of Propp's *story functions*. We do this as a basis of comparison for when we later redefine the story in terms of *tropes*. For this section, the aim of which is to demonstrate the use of *logic* for narrative rather than of tropes, Propp's formalism is used for simplicity and demonstrative purposes.

(1)

(2)

initial?

explain limitations rather than leaving them implicit.

nice intro

1.1.1 Introducing "Punch and Judy"

Throughout this thesis we use the *Punch and Judy* story world for our example narratives. *Punch and Judy* is a traditional British-Italian puppet show often seen at seaside resorts. Its first recorded appearance in the UK is in Covent Garden, London in 1662, having been brought into the country by Italian puppeteer Pietro Gimonde. Since then it has become a British cultural institution, delighting children and adults alike over the course of centuries.

The show itself is an incredibly violent farce, where the titular character *Punch* is a homicidal maniac with a hair-trigger temper. Most scenes start with him having an argument with

SW 2.5
2.5

another puppet, and end with Punch beating the other puppet to death. He starts by killing his wife and child, then the policeman that comes to arrest him, before finally murdering the Devil himself.

Interestingly, there is no comeuppance for Punch's killing spree, as most modern audiences would expect. Instead, the story shows Punch as a buffoon, a ridiculous caricature of an angry man who blunders his way through life, yet is victorious nonetheless. The story has no moral message, only farcical violence.

Despite its arguable nihilism, *Punch and Judy* has many features that make it a suitable narrative domain for the purposes of testing our research:

- It has many recognisable patterns, or *tropes*. For example, most scenes feature some kind of audience participation, and most end with slapstick comedy.
- It can be as simple or sophisticated as we choose to make it. Some scenes are relatively simple to model, such as the scene where Punch beats the policeman. Others are more difficult, such as where Punch is asked to guard some sausages, but then loses them to a crocodile. Combining several scenes into a complete and recognisable *Punch and Judy* narrative would be a worthwhile challenge for a narrative generation system.
- Following from this, its scenes can either be self-contained and modelled on their own, or can depend on the context of the previous scenes. For example, Punch's foes increase in importance as his rampage progresses. In some versions, he kills his wife, then a policeman, then the executioner sent to hang him, then finally the Devil. This makes it useful for our modelling of scenes individually, but also as part of a narrative as a whole.
- Most (British) people know Punch and Judy when they see it. They are able to recognise how characters in the story are supposed to behave, and what should happen in each scene and in the story as a whole. This helps with evaluation of the system: the audience will be able to say whether or not the generated tropes are *authentic* or not.

The minimal set of Punch and Judy characters would be Punch, his wife Judy, their baby, Joey the clown (who narrates the story and interacts with the audience, and is immune to Punch's attacks), a crocodile and a Policeman. Extended versions include other animals that Punch chases (such as a monkey or cat), and the previously mentioned hangman and Devil characters. These are not needed for a full Punch and Judy show, however. Indeed, many performers show just one or two scenes in a show, with the first featuring Punch, Judy and the baby, and then finishing with the scene where Punch wrestles with a crocodile for some sausages.

A Punch and Judy Scene Described as Propp Functions

The common elements of Punch and Judy are easily described in terms of Propp's story functions. Here we pick one scene to use as an example: the scene where Punch battles a Crocodile in order to safeguard some sausages. In this scene, Joey the clown (our narrator) asks Punch to guard the sausages. Once Joey has left the stage, a Crocodile appears and eats the sausages. Punch fights with the Crocodile, but it escapes. Joey then returns to find that his sausages are gone. The corresponding story functions are:

1. Joey tells Punch to look after the sausages (*interdiction*).
2. Joey has some reservations, but decides to trust Punch (*complicity*).
3. Joey gives the sausages to Punch (*provision or receipt of a magical agent*).
4. Joey leaves the stage (*absentation*).
5. A Crocodile enters the stage and eats the sausages (*violation*).
6. Punch fights with the Crocodile (*struggle*).
7. Joey returns to find that the sausages are gone (*return*).

Some story functions map to Punch and Judy better than others (for example, it is debatable as to whether or not the sausages can be considered a "magical agent"), but Propp's formalism seems well suited to Punch and Judy for the most part. The advantage of using Propp for the Punch and Judy story domain is that the story function concept is like a ~~very simple~~ finite state machine, which we can easily model in logic.

1.1.2 Propositional and Predicate Logics

1.1.3 Modal Logic and Kripke Structures

Modal logic extends classical propositional and predicate logic with modalities, which are operators that qualify a statement. For example, rather than simply stating 'The Crocodile eats the sausages', we could instead say "the Crocodile sometimes eats the sausages", or "it's possible that the Crocodile eats the sausages". Classic modal logic deals with *alethic modality*, which describes whether a statement is *possible* or *necessary*. This is implemented using unary operators to qualify statements. For example, $\Diamond P$ states that P is possible and $\Box P$ states that P is necessary. Naturally, we can use modalities beyond just possibility and necessity. Any modal logic can be constructed using operators and modalities of the creator's choice.

At the end of this chapter (from section 1.3 onwards), we describe our method of formally constructing narrative and trope descriptions with social institutions and norms. This technique uses ideas from *deontic* logic, a modal logic that adds *permission* and *obligation* operators to logical statements. In order to demonstrate the advantages of describing stories as social institutions, we first use a more straightforward modal logic to describe the events of a story: Interval Temporal Logic (ITL). ITL allows us to formally describe the temporal ordering of events that occur in a story, enabling the use of theorem-proving tools to check whether certain sequences of events are possible. Later, we contrast this temporal-oriented approach to the character and socially-oriented approach of describing stories with institutions.

Temporal Logics Arthur Prior is the first to employ modal logic as a way of describing sequences of time in his 1957 work *Time and Modality* Prior (2003). Here he uses just two modal operators, P and F, representing *some time in the past* and *some time in the future* respectively. Hans Kamp adds two extra operators to Prior's logic, *Since* and *Until*, in his 1968 thesis Kamp (1968), enabling it to describe spans of time in addition to the ordering of temporal events. As Kripke later points out to Prior, this model lacks the expressiveness needed to describe all possible sequences of events. One major shortcoming is its restriction

to describing only *linear* events.

Linear Temporal Logic (LTL), though still limited to the description of linear sequences of events, is an evolution of the work done by Prior and Kamp. Proposed by Amir Pnueli in 1977 Pnueli (1977), its original use is for the formal verification of computer programs. Computational Tree Logic (CTL) Ben-Ari et al. (1983) is similar to LTL, but allows for the representation of non-linear time through the allowance of branches. Through CTL it is possible to describe several alternative pathways through time, though only one may ever be actualised. Like LTL, its original purpose is for formal verification of software, and in model checkers. Both CTL and LTL are subsets of CTL* (Computational Tree Logic) Emerson and Halpern (1986), which can both describe both multiple branches of temporal paths and their durations. CTL* formulae must refer to a specific Kripke structure, however (a description of Kripke structures appears in section 1.2).

Interval Temporal Logic In order to model narratives with modal logic, we employ Interval Temporal Logic (ITL), composed of the temporal intervals defined by Allen Allen (1983) and developed into modal operators by Halpern and Shoham Halpern and Shoham (1991). This allows the expressiveness necessary to describe branching, parallel and nested paths through stories.

In most temporal logics (such as CTL* and its subsets), fixed *time points* without duration are the basic unit of time. However, this can make it difficult to reason about the *duration* of events that occur over a period of time. Temporal Interval Logic tackles this problem through the use of *time intervals* or *periods* as the basic temporal unit.

The version of ITL used here is that described by Della Monica et al. in their overview paper Della Monica et al. (2013). Table 1.1 lists the temporal intervals described by Allen, along with their modal operator equivalents in *Halpern-Shoham logic.*

The operators defined by Halpern and Shoham are (a bar over an operator denotes its inverse):

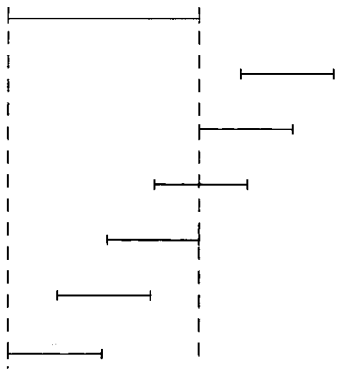
- $\langle L \rangle / \langle \overline{L} \rangle$ (Later): The interval occurs at some point after another interval.
- $\langle A \rangle / \langle \overline{A} \rangle$ (After): The interval occurs immediately after another interval.
- $\langle O \rangle / \langle \overline{O} \rangle$ (Overlaps): The interval occurs both during and before or after another interval.
- $\langle E \rangle / \langle \overline{E} \rangle$ (Ends): The interval ends at exactly the same time as another interval.
- $\langle D \rangle / \langle \overline{D} \rangle$ (During): The interval both starts and ends inside the duration of another interval.
- $\langle B \rangle / \langle \overline{B} \rangle$ (Begins): The interval begins at exactly the same time as another interval.

1.1.4 Propp Example with Punch and Judy

In this example, we combine Halpern and Shoham's temporal operators with the possibility ($\Diamond$) and necessity ($\Box$) operators of modal logic. We follow the convention of writing possibility operators inside angle brackets: $\langle\,\rangle$ and necessity operators within square brackets: $[\,]$.

This example shows the "sausages" scene described in section 1.1.1, consisting of a set of situations S, containing Propp story functions P. The interval temporal logic operators used

Table 1.1: Operators in the Interval Temporal Logic (adapted from Della Monica et al. (2013))

Interval	**Allen notation**	**HS notation**
	*equals* $\{=\}$	
	before $\{<\}$ / *after* $\{>\}$	$\langle L\rangle$ / $\langle\overline{L}\rangle$ *(Later)*
	meets $\{m\}$ / *met-by* $\{mi\}$	$\langle A\rangle$ / $\langle\overline{A}\rangle$ *(After)*
	overlaps $\{o\}$ / *overlapped-by* $\{oi\}$	$\langle O\rangle$ / $\langle\overline{O}\rangle$ *(Overlaps)*
	finished-by $\{fi\}$ / *finishes* $\{f\}$	$\langle E\rangle$ / $\langle\overline{E}\rangle$ *(Ends)*
	contains $\{di\}$ / *during* $\{d\}$	$\langle D\rangle$ / $\langle\overline{D}\rangle$ *(During)*
	started-by $\{si\}$ / *starts* $\{s\}$	$\langle B\rangle$ / $\langle\overline{B}\rangle$ *(Begins)*

$$S = \{S_0, S_1, S_2, S_{3a}, S_{3a_1}, S_4, S_{3b}, S_{3b_1}, S_4, S_5\} \tag{1.1}$$

$$P = \{\texttt{interdiction(A,B,C)}, \texttt{absentation(A)}, \texttt{struggle(A,B)}, \texttt{victory(A)}, \texttt{villainy(A,B)}, \texttt{violation(A,B)}, \texttt{return(A)}\} \tag{1.2}$$

$$T = \{D, \overline{D}, O, \overline{O}, A, \overline{A}, B, \overline{B}, L, \overline{L}, E, \overline{E}\} \tag{1.3}$$

Figure 1-1: Modal operators

in this example are the set T. Figure 1-1 shows the modal operators we use. A, B and C in formula 1.2 are variables that represent the characters and objects that appear in the story. Hybrid logics allow worlds, time intervals, to be named. The hybrid logic nominal operator allows specific times to be uniquely referenced, allowing a logic to talk about specific states such as $S_0..S_5$. The nominal proposition is true of one specific time interval such that any two worlds with the same name represent co-extensive intervals of time. Anything true of one is true of the other. We also make use of a nominal modal operator so that it is possible to make assertions about these named worlds, "It is necessary that in state S_1 Joey absents himself." This technique enables us to associate story functions with specific, named intervals. We use hybrid logic to identify nodes using the *nominal* operator, shown as @. The nominal operator adds the capability of referring to possible worlds in formulas. In this way, each possible state of a system can be labelled and referenced from other states. As seen in figure 1-4, each nominal node must have its own name as a relation leading back to the root node, in order to be linked to and referred from the other nodes. We can combine this with the Interval Temporal Logic to make statements such as "An absentation starts with state $@S_1$ and ends with state $@S_5$." (formula 1.19). Figure 1-3 describes the full sausages scene from Punch and Judy using the time intervals shown in figure 1-2.

One notable feature of this approach is that it enables the building of reusable story components. For example, we have said that an interdiction must begin with an absentation

Figure 1-2: Timings of the story functions in the sausages scene

$$S_0 \wedge interdiction(Joey, Punch, Sausages) \wedge \langle B\rangle @S_1 \wedge \langle E\rangle @S_4 \wedge \langle A\rangle @S_5 \quad (1.4)$$
$$[@S_1]absentation(Joey) \wedge \langle A\rangle @S_2 \quad (1.5)$$
$$[@S_2]struggle(Punch, Crocodile) \wedge \langle E\rangle (@S_{3a} \vee @S_{3b}) \quad (1.6)$$
$$[@S_{3a}]victory(Crocodile) \wedge \langle A\rangle @S_{3a_1} \quad (1.7)$$
$$[@S_{3a_1}]villainy(Crocodile, Sausages) \wedge \langle E\rangle @S_4 \quad (1.8)$$
$$[@S_{3b}]victory(Punch) \wedge \langle A\rangle @S_{3b_1} \quad (1.9)$$
$$[@S_{3b_1}]villainy(Punch, Sausages) \wedge \langle E\rangle @S_4 \quad (1.10)$$
$$[@S_4]violation(Punch, Sausages) \quad (1.11)$$
$$[@S_5]return(Joey) \quad (1.12)$$

Figure 1-3: Sausages scene with nominals and Interval Temporal Logic

and ends with a violation (formula 1.4). This pattern can be reused in different stories, or several times in the same story.

1.2 Describing Punch and Judy with Kripke Structures

We use Kripke structures Kripke (1963) as a method of interpreting the combination of modal logic with Interval Temporal Logic. A Kripke structure is a graph, the nodes of which represent a possible world consisting of a set of assertions, and the edges of which are the accessibility relations between worlds.

1.2.1 LoTREC

In order to build and visualise the Kripke structures, we use LoTREC del Cerro et al. (2001), a generic tableaux prover for modal and description logics. It allows the user to build up Kripke models using a domain specific language and display those models in the form of a

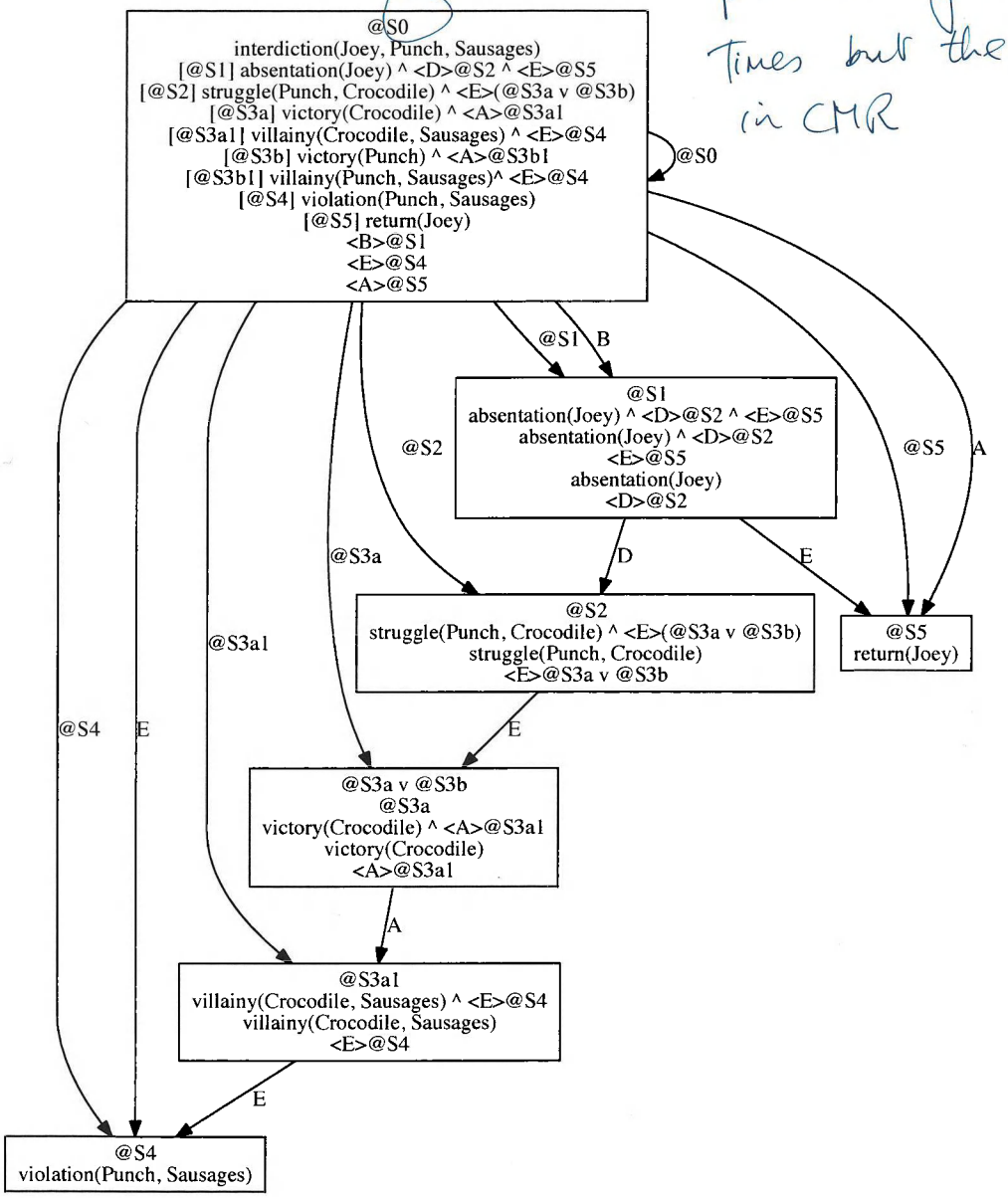


Figure 1-4: One model from the sausages scene in LoTREC

graph diagram. LoTREC uses the tableau method for model checking. This method checks whether or not its input is satisfiable by attempting to build a model for it. If the model construction fails, then the input is unsatisfiable. Building models in LoTREC consists of defining *connectors*, *rules* and *strategies*. Connectors are the logical operators used in formulas, rules are the instructions LoTREC needs to expand them into new nodes and edges and strategies are ways of combining rules in order to form models. LoTREC takes an initial set of formulas as input and expands each formula into its components. These formulas are input using a simple prefix notation domain specific language consisting of operators and arguments, described in the LoTREC instruction book *Kripke's Worlds* Gasquet et al. (2013).

In the expansion, if a [] (necessary) operator is encountered, then the formulas that serve as its argument are propagated across to all subsequent nodes. In the case of a $\langle\rangle$ (possibility) operator, a new node (possible world) is created containing its arguments. In both cases, binary operators can be used where one argument is located inside the square or angle brackets. In these cases, the argument inside the brackets is the accessibility relation, and is used to label the edges leading to the subsequent nodes. In the case of a disjunction ($\vee$) operator, the current model is copied in its entirety and extended with one of the operator's arguments. The other argument is added onto the current model. This is an important way of creating alternative routes through the narrative, all of which can be checked for consistency.

The Sausages Scene in LoTREC The narrative is captured as a system of interval temporal logic assertions over story functions. The assertions are interpreted using a tableaux reasoner by unpacking them to form a Kripke structure. Using the initial formulas from figure 1-3 as input, figure 1-4 shows the model for the case where Punch wins the fight with the Crocodile. One other model exists in this scenario, in which the Crocodile is instead the victor.

The example in figure 1-4 describes the fabula of a branching story, where either Punch or the Crocodile may win the fight for the sausages. This corresponds to the disjunction in figure 1-3, formula 1.6. This leads to the creation of two models: the one in which the Crocodile wins and then goes on to eat the sausages (situation $@S_{3a}$), and the one in which Punch wins (situation $@S_{3b}$). Using a breadth-first strategy like LoTREC we would build all models simultaneously. This enables them to be checked for internal consistency, eliminating potentially impossible temporal arrangements. For example, no interval may come after or later than itself.

The first node contains the logical description of the story, which is then unpacked into subsequent nodes. It starts with the initial situation, S_0, which contains an interdiction: $interdiction(Joey, Punch, Sausages)$, where Joey tells Punch to look after the sausages. The other situations, S_1 to S_5 are listed using the necessity operator, with their accessibility relations being each situation's nominal operator (such as $@S_1$). This means that these situations are all created as new nodes, linked to the root node using their nominal accessibility relation, and so can be referred to later using the nominal operator. In this way, situations can refer to other situations.

For example, in Propp's formalism, an interdiction must begin with the absentation of a character and end with the interdiction's violation. For this reason, the initial situation is connected to $@S_1$, $(absentation(Joey))$, with the $\langle B\rangle$ (begins) accessibility relation and to $@S_4$, $(violation(Punch, Sausages))$, with the $\langle E\rangle$ (ends) accessibility relation. As other situations are unpacked into nodes, they are linked to other situations in the same way. An example of this is shown in S_1, the absentation, which must end with Joey's return, S_5, and during which a struggle must occur (S_2). As the formulas inside S_1 are unpacked, S_1 is linked to the other situations with the appropriate temporal accessibility relations. This is all made possible with the hybrid logic, which allows nodes to be referred to by name and linked to.

By describing situations that are linked together with temporal relations, we can build narratives and check them as we go. If a narrative is inconsistent in some way (by stating that a character is both dead and alive, for example), then the model will be unsatisfiable. This is useful for ensuring the construction of consistent narratives.

Additionally, every time a story has the possibility of branching off, the model for each new branch can be checked to see if it is viable in the narrative world. This means that rather than relying on an author to write out each branch of a story, they can be generated automatically from a story description and checked for inconsistencies.

As mentioned previously, the combination of Interval Temporal Logic and a narrative formalism such as Propp's allows us to create abstracted story components. For example, we have described the interdiction as being started by an absentation and ended with a violation. This pattern can be called the "interdiction pattern", and so can be easily reused in other narratives.

The use of LoTREC has shown the utility of being able to visualise a branching narrative in its entirety. This suggests that our method enables the visual authoring of interactive narratives by non-technical creators. Rather than having to type in computer code, or logical formulas, a visual tool could be developed to create logically consistent narrative worlds.

This approach for describing narrative has many potential real-world uses. One obvious use would be for computer game narratives, but it could also be used to describe a repeatable training scenario, and the possible choices available to a user at any point during the simulation.

1.3 Norms and Institutions

While the use of Interval Temporal Logic and model checking techniques is useful for the creation and checking of paths through a story, its main limitation is its unsuitability for use in a multi-agent system (MAS). Similar to the planner-based systems described in section **??**, its implementation in a MAS would involve the use of a "director", which tells each agent what to do during the course of the story. This greatly reduces the flexibility of using a MAS for interactive narrative. What advantage is there of using intelligent agents as characters in a story world if they have no agency to make use of their intelligence? By contrast, modelling stories with social norms and institutions offers a more flexible way to encourage agents actions to conform to the shape of a story.

An institution describes a set of 'social' norms describing the permitted and obliged behaviour of interacting agents. Noriega's 'Fish Market' thesis Blanco-Vigil (1998) describe how an institutional model can be used to regiment the actions of agents in a fish market auction. Several Artikis et al. (2009); Fornara et al. (2007); Cardoso and Oliveira (2007) extend this idea to build systems where institutions actively regulate the actions of agents, while still allowing them to decide what to do. We build on the work of Cliffe et al. Cliffe et al. (2007) and Lee et al. Lee et al. (2013) to adapt it for the world of narrative, using an institutional model to describe the story world of Punch and Judy in terms of Propp's story moves and

character roles, through which the actors acquire powers and permissions appropriate to the character and the story function in which they are participating.

Institutional models use concepts from deontic logic to provide obligations and permissions that act on interacting agents in an environment. By combining this approach with Propp's concepts of *roles* and *story moves*, we describe a Propp-style formalism of Punch and Judy in terms of what agents are *obliged* and *permitted* to do at certain points in the story.

For example, in one Punch and Judy scene, a policeman enters the stage and attempts to apprehend Punch. According to the rules of the Punch and Judy world, Punch has an obligation to kill the policeman by the end of the scene (as this is what the audience expects to happen, having seen other Punch and Judy shows). The policeman has an obligation to try his best to catch Punch. Both agents have permission to be on the stage during the scene. The policeman only has permission to chase Punch if he can see him (Punch is obliged to hide from him at the start of the scene).

The permissions an agent has, on the one hand, constrain the choices of actions available to them at any given moment. Obligations, on the other hand, affect the goals of an agent. Whether or not an agent actively tries to fulfil an obligation depends on their emotional state.

1.3.1 Institution example

To illustrate the application of institutional modelling, we here continue the 'sausages and crocodile' scene example from section 1.1.1, taking the Propp story functions and describing them in an institutional model. We define our institution in terms of *fluents*, *events*, *powers*, *permissions* and *obligations*, following Cliffe et al. (2007), to which the interested reader is referred for the full details of the formal model, including the generate ($\mathcal{G}$) and consequence ($\mathcal{C}$) relations, which are only described here in sufficient depth for the model being presented.

Fluents

These are properties that may or may not hold true at some instant in time, and that change over the course of time. *Institutional events* are able to *initiate* or *terminate* fluents at points in time. A fluent could describe whether a character is currently on stage, the scene of the story that is currently being acted out, or whether or not the character is happy at that moment in time. Domain fluents ($\mathcal{D}$) describe domain-specific properties that can hold at a certain point in time. In the Punch and Judy domain, these can be whether or not an agent is on stage, or their role in the narrative:

$$\mathcal{D} = \{\texttt{onstage}, \texttt{hero}, \texttt{villain}, \texttt{victim}, \texttt{donor}, \texttt{item}\}$$

Institutional fluents consist of (institutional) *powers*, *permissions* and *obligations*. An **institutional power** ($\mathcal{W}$) describes whether or not an external event has the authority to generate a meaningful institutional event. Taking an example from Propp's formalism, an *absentation* event can only be generated by an external event brought about by a *donor* character (such as their leaving the stage). Therefore, any characters other than the donor

character would not have the institutional power to generate an *absentation* institutional event when they leave the stage. The possible empowerments (institutional events) from Propp used in Punch and Judy are:

$$\mathcal{W} = \{\texttt{pow(introduction)}, \texttt{pow(interdiction)}, \texttt{pow(give)}, \\ \texttt{pow(absentation)}, \texttt{pow(violation)}, \texttt{pow(return)}\}$$

Permissions

($\mathcal{P}$) are associated with external actions that agents are permitted to do at a certain instant in time. These can be thought of as the set of *socially permitted* actions available to an agent. While it is possible for an agent to perform other actions, societal norms usually discourage them from doing so. For example, it would not make sense in the world of Punch and Judy if Punch were to give the sausages to the Policeman. It is always Joey who gives the sausages to Punch. Also, it would be strange if Joey were to do this in the middle of a scene where Punch and Judy are arguing. We make sure agents' actions are governed so as to allow them only a certain subset of permitted actions at any one time. The set of permission fluents is:

$$\mathcal{P} = \{\texttt{perm(leavestage)}, \texttt{perm(enterstage)}, \texttt{perm(die)}, \texttt{perm(kill)}, \\ \texttt{perm(hit)}, \texttt{perm(give)}, \texttt{perm(fight)}\}$$

Obligations

($\mathcal{O}$) are institutional facts that contain actions agents *should* do before a certain deadline. If the action is not performed in time, a *violation event* is triggered, which may result in a penalty being incurred. While an agent may be obliged to perform an action, it is entirely their choice whether or not they actually do so. They must weigh up whether or not pursuing other courses of action is worth accepting the penalty that an unfulfilled obligation brings.

Anybody who has seen a Punch and Judy show knows that at some point Joey tells Punch to guard some sausages, before disappearing offstage. Joey's departure is modelled in the institution as the *absentation* event. It could also be said that Joey has an obligation to leave the stage as part of the *absentation* event, otherwise the story function is violated. This can be described in the institution as:

$$\mathcal{O} = \{\text{obl}(\texttt{leavestage}, \texttt{absentation}, \texttt{viol(absentation)})\}$$

The first argument is the external event that must be triggered according to the obligation, the second argument is the institutional deadline event, and the third argument is the violation event which is triggered if the obligation is not fulfilled before the deadline.

Events

Cliffe's model specifies three types of **event**: *external events* (or 'observed events', $\mathcal{E}_{obs}$), *institutional events* ($\mathcal{E}_{instevent}$) and *violation events* ($\mathcal{E}_{viol}$). Examples of each are given in

$$\mathcal{E}_{obs} = \{\texttt{startshow}, \texttt{leavestage}, \texttt{enterstage}, \texttt{die}, \texttt{give}, \texttt{harmed}, \texttt{hit}, \texttt{fight}, \texttt{kill}, \texttt{escape}\} \quad (1.13)$$

$$\mathcal{E}_{instevent} = \{\texttt{introduction}, \texttt{interdiction}, \texttt{receipt}, \texttt{absentation}, \texttt{violation}, \texttt{return}, \texttt{struggle}, \texttt{defeat}, \texttt{complicity}, \texttt{victory}, \texttt{escape}\} \quad (1.14)$$

$$\mathcal{E}_{viol} = \{\texttt{viol(introduction)}, \texttt{viol(interdiction)}, \texttt{viol(receipt)}, \texttt{viol(absentation)}, \texttt{viol(violation)}, \texttt{viol(return)}, \texttt{viol(struggle)}, \texttt{viol(defeat)}, \texttt{viol(complicity)} \texttt{viol(victory)}, \texttt{viol(escape)}\} \quad (1.15)$$

Figure 1-5: External, institutional and violation events for Punch and Judy

Figure 1-5. *External events* are observed to happen in the agents' environment, which can *generate institutional events* which occur only within the institional model, leading to the *initiation* or *termination* of (domain) fluents, permissions, obligations or institutional powers. An external event could be an agent leaving the stage, an agent hitting another, or an agent dying. Internal events include narrative events such as scene changes, or the triggering of Propp story functions such as *absentation* or *interdiction* (described in section **??**). *Violation* is the name of a Propp story function, and is included as an internal event, although it has no relation to the violation events of an institution. Violation events occur when an agent has failed to fulfil an obligation before the specified deadline. These can be implemented in the form of a penalty, by decreasing an agent's health, for example.

Event Generation and Consequences

An **event generation** function, $\mathcal{G}$, describes how events ($\mathcal{E}$, usually external, but can also be internal) can generate other (usually institutional) events, conditional upon the current institutional state ($\mathcal{X}$). This is the counts-as relation. For example, if an agent leaves the stage while the *interdiction* event holds, they trigger the *leavestage* event. This combination generates the *absentation* institutional event (rule 1.19). Further examples appear in figure 1-6.

Event generation functions follow a $\langle\texttt{preconditions}\rangle \rightarrow \{\texttt{postconditions}\}$ format. The preconditions consist of a set of fluents that hold at that time, along with an event to have occurred. The postconditions are the events that are generated. The generation functions are used to generate internal, institutional events from external events.

Consider the Punch and Judy scenario described in section 1.1.1. There are seven institutional events (story functions) that occur during this scene: *interdiction*, *complicity*, *receipt* (from Propp's *receipt of a magical agent*) *absentation*, *violation*, *struggle*, *return*. These institutional events are all generated by external events. The *interdiction* is generated when Joey tells Punch to protect the sausages. Punch agreeing amounts to *complicity*. Joey *gives* punch the sausages (*receipt*), then leaves the stage (*absentation*). The crocodile eating the sausages is a *violation* of Punch's oath, the agents fight (*struggle*), then Joey enters the stage again (*return*).

$$\mathcal{G}(\mathcal{X},\mathcal{E}):\begin{cases} \langle \emptyset, \mathit{tellprotect}(\texttt{donor},\texttt{villain},\texttt{item})\rangle \rightarrow \{\mathit{interdiction}\} & (1.16)\\ \langle \{\mathit{interdiction}\}, \mathit{agree}(\texttt{villain})\rangle \rightarrow \{\mathit{complicity}\} & (1.17)\\ \langle \emptyset, \mathit{give}(\texttt{donor},\texttt{villain},\texttt{item})\rangle \rightarrow \{\mathit{receipt}\} & (1.18)\\ \langle \{\mathit{interdiction}\}, \mathit{leavestage}(\texttt{donor})\rangle \rightarrow \{\mathit{absentation}\} & (1.19)\\ \langle \{\mathit{interdiction}\}, \mathit{harmed}(\texttt{item})\rangle \rightarrow \{\mathit{violation}\} & (1.20)\\ \langle \{\mathit{interdiction}, \mathit{absentation}\}, \mathit{enterstage}(\texttt{donor})\rangle \rightarrow \{\mathit{return}\} & (1.21)\\ \langle \emptyset, \mathit{hit}(\texttt{donor},\texttt{villain})\rangle \rightarrow \{\mathit{struggle}\} & (1.22) \end{cases}$$

Figure 1-6: Event generation in the sausage scene

It is desirable that these story functions occur in this sequence in order for a satisfying narrative to emerge. Agents may decide to perform actions that diverge from this set of events, but the institution is guiding them towards the most fitting outcome for a *Punch and Judy* world. For this reason, a currently active story function can be the precondition for event generation. For example, the *receipt* event may only be triggered if an agent externally performs a *give* action **and** if the *complicity* event currently holds (rule 1.18). Examples of event generation function for this scenario, complete with preconditions, are listed in rules 1.16–1.22 (Figure 1-6).

Consequences consist of fluents, permissions and obligations that are *initiated* ($\mathcal{C}^{\uparrow}$) or *terminated* ($\mathcal{C}^{\downarrow}$) by institutional events. For example, the institutional event *receipt* initiates the donor agent's permission to leave the stage, triggering the *absentation* event (rule 1.24). When the *interdiction* event is currently active and a *violation* event occurs, the interdiction event is terminated (1.31). Rules 1.23–1.32 in Figures 1-7 and 1-8 describe the initiation and termination of fluents in the Punch and Judy sausages scene detailed in section 1.1.1.

Now that we have created the Punch and Judy story institution, the next section goes through the steps required to incorporate this institution into a working, interactive, Punch and Judy puppet show simulation using a multi-agent system.

1.4 An Interactive, Generative Punch and Judy Show with Institutions

Our main criticism of the use of Interval Temporal Logic for the description of story worlds is its lack of flexibility in governing the actions of intelligent agents in a multi-agent system. If agents are told precisely what to do in a story environment, there is little reason in giving them intelligence. We demonstrate this point further by modelling the characters in our Punch and Judy simulation with emotional models that affect their decision making and behaviour. We choose to do this so that agents would be able to deviate from the narrative in times of extreme emotion, providing some level of unpredictability to the story.

This section describes a fully realised and implemented interactive puppet show built with institutions and emotional agents. In this show, the characters interact on screen in accordance with the Punch and Judy story world. At certain points in the show, the audience are encouraged to cheer or boo at the characters. When this happens, a character puppet's

$$
\mathcal{C}^{\uparrow}(\mathcal{X},\mathcal{E}):\begin{cases}
\langle \emptyset, \texttt{interdiction}\rangle \rightarrow \{\text{active}(\texttt{interdiction}), \\
\qquad\qquad \text{perm}(\texttt{give}(\texttt{donor},\texttt{villain},\texttt{item}))\} & (1.23)\\
\langle \emptyset, \texttt{receipt}\rangle \rightarrow \{\text{perm}(\texttt{leavestage}(\texttt{donor}))\} & (1.24)\\
\langle \{active(absentation)\}, \\
\texttt{enterstage}(\texttt{villain})\rangle \rightarrow \{\text{obl}(\texttt{eat}(\texttt{villain},\texttt{sausages}), \\
\qquad\qquad \texttt{return},\texttt{viol}(\texttt{violation}))\} & (1.25)\\
\langle \{active(interdiction)\}, \\
\texttt{leavestage}(\texttt{donor})\rangle \rightarrow \{\text{obl}(\texttt{enterstage}(\texttt{donor}), \\
\qquad\qquad \texttt{eat}(\texttt{villain},\texttt{sausages}), \\
\qquad\qquad \texttt{viol}(\texttt{return}))\} & (1.26)\\
\{active(interdiction)\}, \\
\texttt{violation}\rangle \rightarrow \{\text{perm}(\texttt{enterstage}(\texttt{dispatcher}))\} & (1.27)\\
\langle \{active(absentation), \\
active(violation)\}, \\
\texttt{return}\rangle \rightarrow \{\text{perm}(\texttt{hit}(\texttt{donor},\texttt{villain}))\} & (1.28)
\end{cases}
$$

Figure 1-7: Fluent initiation in the sausage scene

$$
\mathcal{C}^{\downarrow}(\mathcal{X},\mathcal{E}):\begin{cases}
\langle \emptyset, \texttt{interdiction}\rangle \rightarrow \{\text{perm}(\texttt{give}(\texttt{donor},\texttt{villain},\texttt{item}))\} & (1.29)\\
\langle \{active(interdiction)\}, \\
\texttt{absentation}\rangle \rightarrow \{\text{perm}(\texttt{leavestage}(\texttt{donor}))\} & (1.30)\\
\langle \{active(interdiction)\}, \\
\texttt{violation}\rangle \rightarrow \{active(interdiction)\} & (1.31)\\
\langle \{active(absentation), \\
active(violation)\}, \\
\texttt{return}\rangle \rightarrow \{active(absentation)\} & (1.32)
\end{cases}
$$

Figure 1-8: Fluent termination in the sausage scene

emotional state is changed in accordance with the audience response.

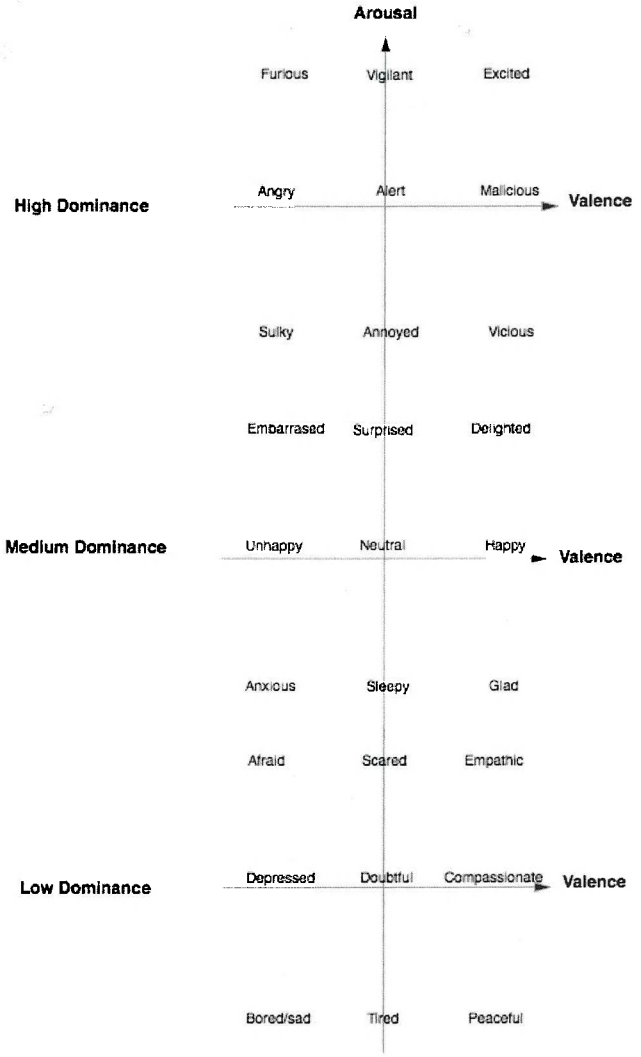
The system was demonstrated at the CDE (Center for Digital Entertainment) and the AISB (Society for the Study of Artificial Intelligence and the Simulation of Behaviour) conferences in 2015, with live audiences interacting with and changing the narrative.

1.4.1 VAD emotional model

In order to make the agents acting out the Punch and Judy show more believable, we apply an emotional model to affect their actions and decisions. For this, we use the valence-arousal (circumplex) model first described by Russell Russell (1980). To give each character its own distinct personality, we extend this model with an extra dimension: dominance, as used by Ahn et al. Ahn et al. (2012) in their model for conversational virtual humans. This dominance level is affected by the reactions of the audience to the agents' actions. For example, Judy may become more dominant as her suggestions to hit Punch with a stick are cheered on by the audience, emboldening her into acting out her impulses. A detailed description appears in the text in Figure 1-9.

It is important to note that the emotional model is part of the agent belief state, and not held in the institution. We want to explore how the characters of the story might be able

I don't think this needs to be here: it can go in the main text and the diagram scaled up (for legibility)



The VAD model, an extension of the Circumplex model of emotion described in section ??, illustrates how valence, arousal and dominance values map to identifiable emotions. Valence, arousal and dominance can each have a value of low, medium or high. This allows the agents to have a total of 27 distinct emotional states. The valence and arousal levels of each agent are affected by the actions of other agents. For example, a character being chased around the stage by Punch will see their valence level drop while their arousal increases. According to Russell's circumplex model of emotion Russell (1980), this would result in them becoming *afraid* (if their dominance level is low).

An agent's emotional state affects its ability to fulfil its institutional obligations. An agent that is *furious* might have no problem carrying out an obligation that requires them to kill another agent. If that same agent is *happy* or *depressed*, however, they might not have the appropriate motivation to perform such a violent action.

Figure 1-9: VAD emotional values (figure adapted from Ahn et al.) Ahn et al. (2012)

to choose actions based on their emotional state. While the institution could theoretically calculate the emotional state for each agent in turn and dictate this to them along with the norms of the narrative, it makes sense to decouple this feature from the narrative institution in order to separate the characters from the events of the story.

Agents' emotional states change according to their interactions with the audience. This is unrelated to what is happening in the narrative, and so this underscores the decision not to include any emotional modelling in the institution. Also, we want the agents to have some degree of freedom within the narrative world. They should be allowed to determine their emotions themselves, so that in extreme emotional states they can perform 'irrational' or 'extreme' actions that may not necessarily fit into the narrative.

Listing 1.1: Emotional rules for a character with medium dominance

```
1  emotion(sleepy) :- valence(0) & arousal(-1) & dominance(0).
2  emotion(neutral) :- valence(0) & arousal(0) & dominance(0).
3  emotion(surprised) :- valence(0) & arousal(1) & dominance(0).
4  emotion(anxious) :- valence(-1) & arousal(-1) & dominance(0).
5  emotion(unhappy) :- valence(-1) & arousal(0) & dominance(0).
6  emotion(embarrassed) :- valence(-1) & arousal(1) & dominance(0).
7  emotion(glad) :- valence(1) & arousal(-1) & dominance(0).
8  emotion(happy) :- valence(1) & arousal(0) & dominance(0).
9  emotion(delighted) :- valence(1) & arousal(1) & dominance(0).
```

Listing 1.2: AgentSpeak rules for changing an agent's emotional values from audience responses

```
1  +!changeMood
2    <- ?emotion(Z);
3       emotion(Z).
4  +response(_, boo) : asking
5    <- -+valence(-1);
6       -+dominance(-1);
7       !changeMood.
8  +response(_, cheer) : asking
9    <- -+valence(1);
10      -+dominance(1);
11      !changeMood.
```

1.4.2 VAD emotions in Jason

Emotions are implemented as beliefs inside an agent. An agent believes it has a certain level of valence, arousal and dominance, and it works out its emotional state based on a combination of these three factors. When the audience cheers or boos them, this changes the belief holding the relevant emotional variable, and their emotional state as a whole is recalculated.

Valence, arousal and dominance values can take values of -1 (low), 0 (medium) or 1 (high). Listing 1.1 shows the emotional belief rules for an agent with medium dominance (a dominance level of 0). Note that an agent maintains beliefs about both its current emotion label (such as sleepy or happy) and the separate valence, arousal and dominance values at the same time. Similar sets of rules handle the belief emotion for the other dominance levels.

Every time an emotional variable (valence, arousal, or dominance) changes, an agent's emotion is changed according to the rules in listing 1.1. While an agent's valence, arousal and dominance belief values affect the way it makes decisions internally, the results of combinations of these values (sleepy, happy, etc) are broadcast as external actions. The reason for this is that an agent's emotional state may affect the way in which the character is animated: changing the speed at which they move or turning their smile into a frown, for example. For this reason, whenever an emotional change takes place, the new emotion is published as an external action of the agent so that observing entities may perceive it. The Bath sensor framework described in section 1.4.4 provides the means for this evidence of the agent's internal state change to be received by the animation system and reflected accordingly in the display.

Listing 1.2 shows the AgentSpeak rules describing how an agent's valence and dominance levels are changed by the audience cheering or booing their actions. These AgentSpeak plans describe what the agent should do in response to a goal addition (denoted by a `+!` at the start

refer to line numbers. You can use mathescape to embed $\label{—}$ on a line

of the plan name) or a belief addition (prefixed by a simple +). In the case of Listing 1.2, the `+!changeMood` plan updates the agent's emotional state based on its valence-arousal-dominance values and broadcasts the result as an external action. The `+response` plans raise or lower an agent's valence and dominance levels depending on whether the agent perceives a "boo" or "cheer" response from the audience.

An agent announces what they intend to do, then waits three seconds. During this time, they have the belief that they are 'asking' the audience, and listen for a response. A boo reduces an agent's valence and dominance, while a cheer raises them. For each response, the `changeMood` goal is triggered, which looks up and broadcasts the agent's emotional state to the other agents and environment.

1.4.3 Agent decision making

The agents choose which goals to pursue according to three factors: their permitted actions, their obliged actions and their emotional state. Though obliged actions are given priority, and while agents' decisions are generally constrained by their permitted actions, an agent's emotional state has the final say in its decisions. In this way, an agent will follow the social norms of the narrative, but only according to their own mood.

Agent goals and plans The agents are implemented using a belief-desire-intention (BDI) psychological model using the Jason platform (Bordini et al. (2007). An agent's knowledge about the state of their world and themselves are stored as *beliefs*, with new information coming in from the environment getting added to their belief base as *percepts*, which are ephemeral and only last for one reasoning cycle of an agent.

Agents are created with goals and plan libraries. Any goal that an agent is set on carrying out at any point is an *intention*, whereas a goal that an agent has but is not yet pursuing is a *desire*. Plan libraries describe the steps agents need to take in order to achieve goals, as well as how to react to changes in agents' environments.

Norms as percepts When an event occurs, it is added to the event timeline, which is used to query the ASP (Answer Set Programming) solver to obtain the set of norms that hold after the new event has occurred. The new permissions and obligations are then added to each agent as *percepts*. Each time this happens, the set of permitted and obliged actions that an agent sees is changed to be only those that apply at that instant in time, with the previous norms being discarded.

Agents choose between permitted and obliged actions based on their emotional state at the point of decision making. Obliged actions are given a higher priority over permitted ones for most of the emotional states that an agent can be in, though not always. If an agent is in a sulky mood, for example, they may decide to ignore what they are obliged to do by the narrative, even though they know there will be consequences.

For example, in the scene where Joey gives the sausages to Punch, Punch may see that he has permission to eat the sausages, drop them, fight the crocodile, run away (leave the stage)

or shout for help at the crocodile or audience. His obligation for the scene, in accordance with the Punch and Judy narrative world, is to either eat the sausages himself, or let the crocodile have them. This ends Propp's *interdiction* story function with a *violation* function. Note that his obligation in this case is not to guard the sausages as asked to by Joey. While Joey's entrusting of the sausages is certainly an obligation in itself, Punch's main obligations are to the narrative. Lesser obligations towards characters in the story can be implemented as having a lower prority than those of the story itself.

Similarly, at times of extreme emotion, an agent may decide to disregard their set of permitted actions entirely, instead acting out their innermost desires. For example, an angry Punch might decide to just attack Joey instead of agreeing to look after the sausages, or he might just decide to give up and leave if he is depressed. The key point is that the norms act as the will of the *narrative*, guiding the story forward, rather than a strict set of rules that the agents must follow at all costs.

Violation events add percepts to the agents telling them that they are in violation of the narrative norms. Once an agent receives such a percept, an emotional variable is changed. Typically, their dominance will decrease. The reasoning behind this is that if agents are unwilling to participate in the story, they should have less influence in its course of events.

1.4.4 Architecture

Multi-Agent System We use the JASON framework for belief-desire-intention (BDI) agents (Bordini et al. (2007)), programming our agents in the AgentSpeak language. The VAD emotional model is represented inside each agent as a set of beliefs. Each agent has beliefs for its *valence*, *arousal* and *dominance* levels, each of which can take the value of low, medium or high, as discussed in section 1.4.1. This combination of VAD values creates one of the 27 emotional states shown in figure 1-9, affecting whether or not an agent breaks from its permitted or obliged behaviour.

Institutional Framework To describe our institutional model, we use InstAL (Cliffe et al. (2007)), a domain-specific (action) language for describing institutions that compiles to AnsProlog, a declarative programming language for Answer Set Programming (ASP). InstAL's semantics are inspired by the Situation Calculus (Reiter (1991)) and the Event Calculus (Kowalski and Sergot (1989)). InstAL describes how external events generate institutional events, which then initiate or terminate fluents that hold at certain points in time. These fluents can include the permissions and obligations that describe what an agent is permitted or obliged to do when, as described in section **??**. For example, if an agent with the role of *dispatcher* leaves the stage, it generates the Propp *absentation* move in the institution, but only if the *interdiction* function is active (i.e., the *activeFunction(interdiction)* fluent holds):

```
1 leaveStage(X) generates intAbsentation(X)
2    if role(X, dispatcher), activeFunction(interdiction);
```

which generates the following AnsProlog code:

Listing 1.3: Encoding of Propp's *absentation* function

```
1 intAbsentation(X) initiates activeFunction(
      absentation);
2 intAbsentation(X) initiates perm(harm(Y, Z)) if
      role(Y, villain), objStage(Z), onStage(Y),
      activeFunction(interdiction);
3 intAbsentation(X) initiates perm(harm(Y, Z)) if
      role(Y, ambusher), objStage(Z), onStage(Y),
      activeFunction(interdiction);
4 intAbsentation(X) initiates perm(enterStage(X)),
      activeFunction(absentation);
5 intAbsentation(X) initiates perm(enterStage(croc))
      if objStage(sausages);
6 intAbsentation(X) terminates onStage(X), perm(
      leaveStage(X));
```

- The *activeFunction(absentation)* function holds after any *intAbsentation* institutional event, indicating that that Propp function is currently underway
- The *absentation* function gives the villain permission to harm an object, if both are on stage and the *interdiction* function is active
- Same as above, but for *ambusher* role
- The absented character has permission the re-enter the stage at any point during the *absentation* function
- The crocodile has permission to enter the stage if the sausages are on stage
- The *absentation* function means that the absented character is no longer on stage, and cannot leave the stage (as they have already done so)

```
1 occurred(intAbsentation(X),pj,I) :-
2    occurred(leaveStage(X),pj,I),
3    holdsat(pow(pj,intAbsentation(X)),pj,I),
4    holdsat(role(X,donor),pj,I),
5    holdsat(activeFunction(interdiction),pj,I),
6    agent(X), inst(pj), instant(I).
```

- The internal *absentation* event occurs if the following conditions are met:
- $\bullet X$ leaves the stage
- $\bullet X$ has the power to leave the stage
- $\bullet X$ has the role of donor
- • the *interdiction* function is active
- $\bullet X$ is an agent, pj is an institution, I is an instant

The *absentation* institutional event gives the crocodile permission to enter the stage if there are any sausages on the stage. It also terminates the permission of the absented agent to leave the stage, as they have already done so:

```
1 intAbsentation(X) initiates perm(enterStage(croc))
      if objStage(sausages);
2 intAbsentation(X) terminates onStage(X), perm(
      leaveStage(X));
```

- The *absentation* function gives the crocodile permission to enter the stage if the sausages are on stage
- The *absentation* function means that once X leaves the stage, they are no longer on stage

which generates the following:

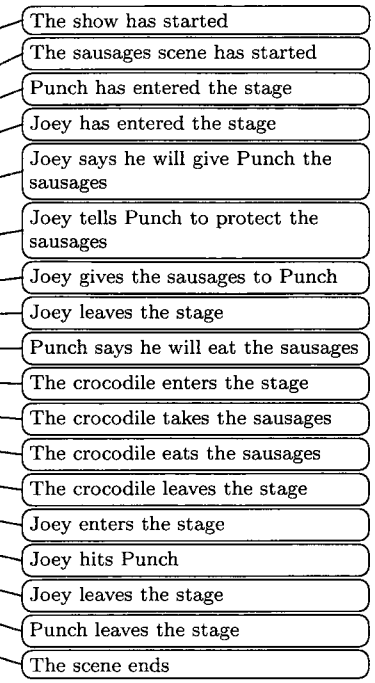
```
1 initiated(perm(enterStage(croc)),pj,I) :-
2    occurred(intAbsentation(X),pj,I),
3    holdsat(live(pj),pj,I), inst(pj),
4    holdsat(objStage(sausages),pj,I),
5    agent(X), inst(pj), instant(I).
```

- The crocodile gets permission to enter the stage if the following conditions are met:
- • the *absentation* function event has occured
- • the pj institution is running
- • the sausages are on stage

By combining statements such as the above, we can build a complete description of the sausages scene in terms of agent norms, such as the Propp *absentation* function, shown in Listing 1.3. InstAL rules like those shown above and in Listing 1.3 are compiled into AnsProlog, then we use the *clingo* answer set solver (Gebser et al. (2011)) to ground the program, and 'solve' queries by finding all permissions and obligations that apply to any agents, given a sequence

Listing 1.4: Possible full trace for sausages scene

```
1  observed(startShow,pj,0).
2  observed(startScene(sausages),pj,1).
3  observed(enterStage(punch),pj,2).
4  observed(enterStage(joey),pj,3).
5  observed(say(joey, give),pj,4).
6  observed(say(joey,  protect),pj,5).
7  observed(give(joey,punch,sausages),pj,6).
8  observed(leaveStage(joey),pj,7).
9  observed(say(punch, harm),pj,8).
10 observed(enterStage(croc),pj,9).
11 observed(take(croc, sausages),pj,10).
12 observed(eat(croc, sausages),pj,11).
13 observed(leaveStage(croc),pj,12).
14 observed(enterStage(joey),pj,13).
15 observed(hit(joey, punch),pj,14).
16 observed(leaveStage(joey),pj,15).
17 observed(leaveStage(punch),pj,16).
18 observed(startScene(end),pj,17).
```



of events as the query input. The agents' percepts are then updated with their permitted and obliged actions from that time instant onwards. Thus, the institutional model acts as a social narrative sensor, interpreting actors' actions in the context of the combination of the concrete narrative and the abstract story moves which detach (instantiate) the norms that guide the actors in the direction of the conclusion of the story arc.

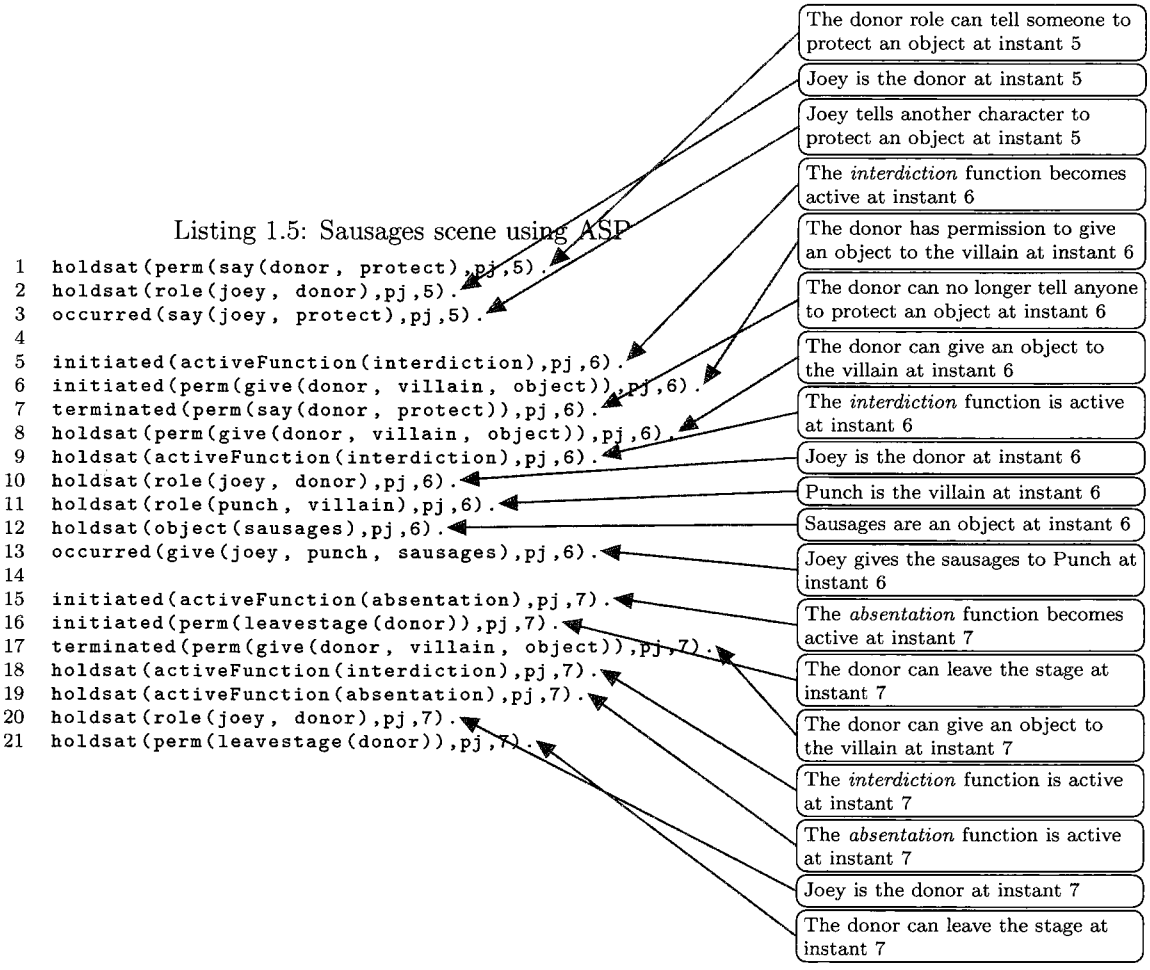
A query is simply a list of external events in chronological order, also called a *trace*. A possible trace describing the actions of agents acting out the *sausages* is described in Listing 1.4. The 'pj' in the trace is the name of the institution that observes the events, while the number is the enumeration of events in the sequence.

Each *observed* event triggers a corresponding *occurs* event inside the institution, as determined by the generates relation. Listing 1.5 shows an extract from an answer set output for the trace queried against the ASP description of the sausages scenario, for events 5 to 7 of the scene. Starting with an initial set of fluents that hold at instant 5, only fluents that have been initiated and not terminated hold at the next instant. For ease of reading, the listing only shows roles that hold at certain instants when they have some effect on the scene, although in practice, all role fluents hold throughout the scene. Figure 1-10 shows a visualisation of the answer set for the trace in listing 1.4.

Bath Sensor Framework The components communicate using the Bath Sensor Framework (BSF) Lee et al. (2013), through publish / subscribe-style communication between distributed software components, in this case connecting intelligent agents with their virtual environments. It currently uses the XMPP publish/subscribe protocol for communication between agents, environment and other software components. Each agent subscribes to receive notifications of environment changes via the appropriate topic node in the XMPP server, which relays messages

Listing 1.5: Sausages scene using ASP

```
1  holdsat(perm(say(donor, protect),pj,5).
2  holdsat(role(joey, donor),pj,5).
3  occurred(say(joey, protect),pj,5).
4
5  initiated(activeFunction(interdiction),pj,6).
6  initiated(perm(give(donor, villain, object)),pj,6).
7  terminated(perm(say(donor, protect)),pj,6).
8  holdsat(perm(give(donor, villain, object)),pj,6).
9  holdsat(activeFunction(interdiction),pj,6).
10 holdsat(role(joey, donor),pj,6).
11 holdsat(role(punch, villain),pj,6).
12 holdsat(object(sausages),pj,6).
13 occurred(give(joey, punch, sausages),pj,6).
14
15 initiated(activeFunction(absentation),pj,7).
16 initiated(perm(leavestage(donor)),pj,7).
17 terminated(perm(give(donor, villain, object)),pj,7).
18 holdsat(activeFunction(interdiction),pj,7).
19 holdsat(activeFunction(absentation),pj,7).
20 holdsat(role(joey, donor),pj,7).
21 holdsat(perm(leavestage(donor)),pj,7).
```



between publishers and subscribers. If any environment change occurs, all subscribed agents are informed of the changes.

Audience Interaction The puppet show is designed to be run in front of either a single user's computer, or on a large display in front of an audience. The user/audience is instructed to cheer or boo the actions of the characters of the show, which is picked up by a microphone and 'heard' by the agents. This then affects the emotional state of the agents and changes the actions they make in the show. Their actions are constrained by the set of 'Punch and Judy' world norms as described in the institutional model.

There are many different ways in which the audience's responses can affect the outcomes of the show. If the audience craves a more 'traditional' Punch and Judy experience, then they can cheer Punch into beating and killing all of his adversaries (including his wife, Judy). Alternatively, a more mischievous audience could goad Judy into killing Punch and then taking over his role as sadist and killer for the rest of the show. The narrative outcomes are dependent on how the audience responds to the action, yet still conform to the rules of the Punch and Judy story world.

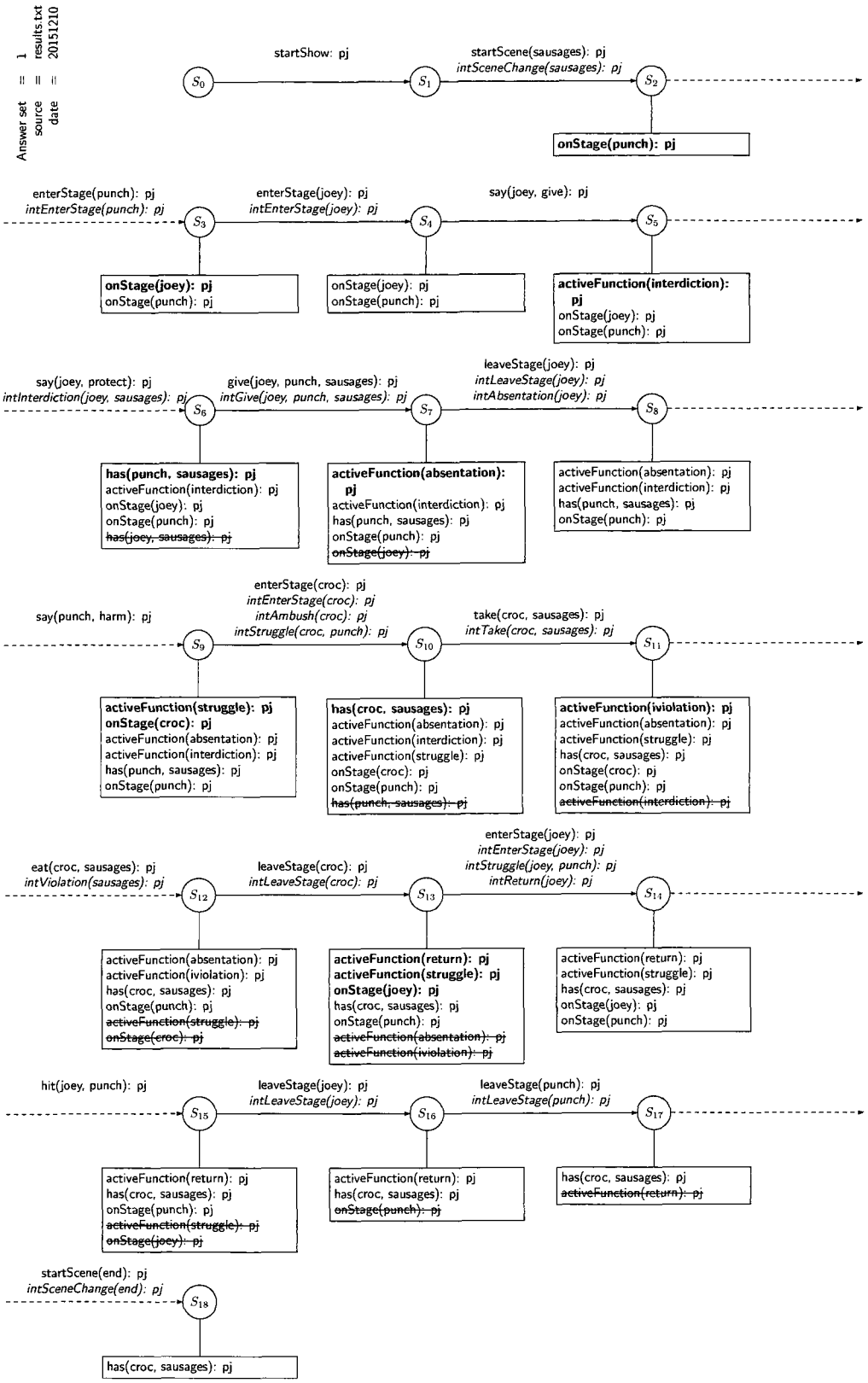
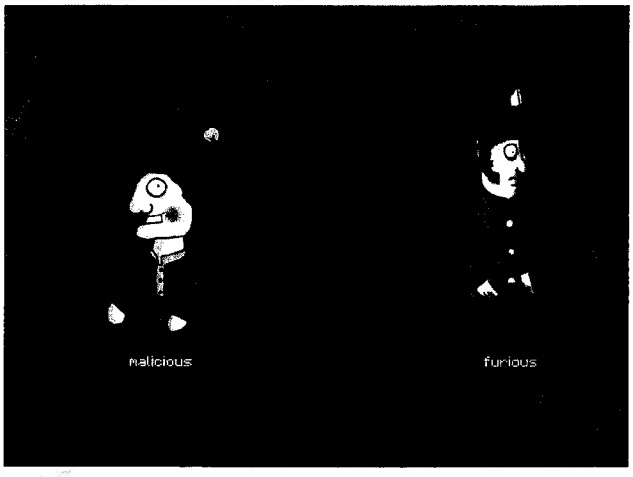


Figure 1-10: Trace visualisation



The animation engine that shows the visual output of the agents actions is written in Javascript and the Phaser game framework. It runs entirely in a browser, and communicates with BSF using the Strophe XMPP library.
If the user allows the program access to their microphone, they can cheer or boo the actions of the agents by shouting into the microphone. Otherwise, they can simulate these actions by clicking on 'cheer' or 'boo' buttons at the bottom of the screen.

Figure 1-11: A screenshot of the Punch and Judy show

1.4.5 Reflections on Describing Narratives with Institutions

Our implementation of the interactive Punch and Judy show demonstrates several advantages in using social institutions to model narrative. Advantages include:

- Because norms encourage rather than enforce actions, agents are given more flexibility in their behaviour. While their actions are encouraged to conform to a narrative for the most part, an author could insert mechanisms by which agents are able to "break away" from the confines of the story. For example, if each agent were to have an emotional model, they could carry out story-disrupting actions in times of extreme emotional distress.
- Rather than thinking about and describing the goals of a story (as with a planner-based system), a narrative is described in terms of what *may* and *must* happen. This gives an author more control over the flexibility of their stories, as they can describe optional paths through a story rather than setting hard goals that must always be satisfied.
- Through the use of InstAL and its compilation to ASP, we can generate all of the possible actions in each story path as answer sets.

However, the major shortcoming of using InstAL to describe stories and tropes in terms of institutions is the learning effort required for a non-programmer (such as a story author) to use it. For this reason, we have developed our own constrained natural-language domain specific language for the description of tropes, which we describe in the next section.

(1) we have not discussed Tropes yet. Perhaps this para is premature?

There is no conclusion to the chapter as a whole. It risks feeling disjointed with the sections about logic (snallish) and then the long section about P.J. I realise you are re-using material but this (lack of) joins has to be handled better and there has to be a coherent overarching narrative for why all this material is here together, beginning with the chapter intro, ending with the chapter summary and bits of glue material in between the sections.

Bibliography

Ahn, J., Gobron, S., Garcia, D., Silvestre, Q., Thalmann, D., and Boulic, R. (2012). An NVC emotional model for conversational virtual humans in a 3D chatting environment. In *Articulated Motion and Deformable Objects*, pages 47–57. Springer.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Artikis, A., Sergot, M., and Pitt, J. (2009). Specifying norm-governed computational societies. *ACM Transactions on Computational Logic (TOCL)*, 10(1):1.

Ben-Ari, M., Pnueli, A., and Manna, Z. (1983). The temporal logic of branching time. *Acta informatica*, 20(3):207–226.

Blanco-Vigil, P. C. N. (1998). *Agent mediated auctions: the fishmarket metaphor.* PhD thesis, Universitat Autònoma de Barcelona.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason.* John Wiley & Sons.

Cardoso, H. L. and Oliveira, E. (2007). Institutional reality and norms: Specifying and monitoring agent organizations. *International Journal of Cooperative Information Systems*, 16(01):67–95.

Cliffe, O., De Vos, M., and Padget, J. (2007). Specifying and reasoning about multiple institutions. In Vázquez-Salceda, J. and Noriega, P., editors, *COIN 2006*, volume 4386 of *Lecture Notes in Computer Science*, pages 63–81. Springer. ISBN: 978-3-540-74457-3. Available via `http://dx.doi.org/10.1007/978-3-540-74459-7_5`.

del Cerro, L. F., Fauthoux, D., Gasquet, O., Herzig, A., Longin, D., and Massacci, F. (2001). Lotrec: the generic tableau prover for modal and description logics. In *Automated Reasoning*, pages 453–458. Springer.

Della Monica, D., Goranko, V., Montanari, A., Sciavicco, G., et al. (2013). Interval temporal logics: a journey. *Bulletin of EATCS*, 3(105).

Emerson, E. A. and Halpern, J. Y. (1986). "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM (JACM)*, 33(1):151–178.

Fornara, N., Viganò, F., and Colombetti, M. (2007). Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, 14(2):121–142.

Gasquet, O., Herzig, A., Said, B., and Schwarzentruber, F. (2013). *Kripke's Worlds: An Introduction to Modal Logics via Tableaux*. Springer Science & Business Media.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124.

Halpern, J. Y. and Shoham, Y. (1991). A propositional modal logic of time intervals. *Journal of the ACM (JACM)*, 38(4):935–962.

Kamp, J. (1968). *Tense logic and the theory of order*. PhD thesis, UCLA.

Kowalski, R. and Sergot, M. (1989). A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer.

Kripke, S. A. (1963). Semantical analysis of modal logic i normal modal propositional calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96.

Lee, J., Baines, V., and Padget, J. (2013). Decoupling cognitive agents and virtual environments. In Dignum, F., Brom, C., Hindriks, K., Beer, M., and Richards, D., editors, *Cognitive Agents for Virtual Environments*, volume 7764 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin Heidelberg.

Pnueli, A. (1977). The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE.

Prior, A. N. (2003). *Time and modality*. Oxford University Press.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 27:359–380.

Russell, J. A. (1980). A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161.