Practical Issues in Detecting Broken Social Commitments

Jason Heard and Rob Kremer

Computer Science Department
University of Calgary
Calgary, Alberta, Canada
{heard, kremer}@cpsc.ucalgary.ca

Abstract. An open system should admit agents from many sources and these agents may have conflicting goals. Therefore, some actions that an agent would like to perform could be detrimental to other agents. Such actions can be either acceptable or unacceptable within a given system. Social norms define what actions are acceptable and unacceptable within a given society. There should be a way to limit the actions of agents to enforce these social norms. One way to begin to accomplish this goal is to have the system observe the actions of agents to model their behaviour. Behaviours that do not conform to specified norms could then be detected, and some action could be taken to prevent agents from performing further actions that violate social norms.

In this paper we discuss the use of social commitments to allow a system to define social norms and detect violations of those norms. Social commitments model an agent's commitments within a society. Some are implied while others are explicitly stated. Our system uses social commitments to define social norms. This paper focusses on the practical requirements that must be met for a system to implement social commitments as a way of defining social norms and detecting violations of those norms. In addition, we give an overview of how our multi-agent system design supports this goal.

1 Introduction

One of the goals of multi-agent systems (MAS) is to achieve synergy between agents. The goal is to accomplish more with a group of agents working together than could be accomplished by all of the agents working individually [1]. In order to do this, agents must be designed so that they can work with other agents. Another goal of multi-agent systems is to admit agents from many sources (or programmers) into the system [2]. These diverse agents may have conflicting goals. It is possible for agents with conflicting goals to work together on portions of their goals (and thereby achieve synergy).

If agents are working on conflicting goals, it may benefit one agent to perform some act that harms another agent. Take, for example, the case of a simple auction. It is generally acceptable to outbid another agent (assuming that you can meet the bid you have given). But it is generally unacceptable for an agent

F. Dignum, R. van Eijk, and R. Flores (Eds.): AC 2005/2006, LNAI 3859, pp. 124–135, 2006.

to state that some resource is worthless, knowing that it is not, so that another agent will bid lower, or not at all on the object. Both actions obviously are detrimental to another agent, but only one would be considered a violation of social norms.

It would be advantageous if the designer of an open system (one that allows agents with conflicting goals to enter it) takes into account these social norms. A system with no checks on norms would not facilitate cooperation, and would not attract many designers or agents to work within it. On the other hand, a system that is too strict would make it difficult to claim that the agents within it are autonomous [3]. The system outlined in this paper does not restrict actions but instead attempts to detect antisocial agents so that they may be avoided as necessary. To create a system with checks on social norms, that system must be able to detect violations. In order to detect norm violations some methodology must be put into place to map social interactions so that norm violations are observable. Social commitments will be used as the criterion to determine if actions conform to social norms within our system. Actions that break social commitments will be considered to be in violation of social norms.

Social commitments model commitments between agents [4,5] at a social level. Social commitments can be used to define societal norms [6], or to formally describe a protocol based on the social commitments implied by that protocol [7]. In order to detect actions that violate social norms as defined above, it is necessary to detect when social commitments are broken. Some work has been done to detect broken commitments [7,8]. Our system differs from previous work in that our system is open and accepts agents that may perform actions the original system designer didn't account for. Once a system detects that an agent has broken a social commitment and has therefore violated a social norm, some actions should be taken to "punish" the responsible agent. These actions (called sanctions) are discussed by Pasquier, Flores, and Chaib-draa [9].

Our system employes the use of a *social commitment observer* agent to detect broken social commitments. The use of special agents to perform monitoring has been done previously with "sentinals" [10]. We propose instead the use of a single agent to perform the monitoring, the use of social commitments as the framework for detecting unacceptable behaviors, and we maintain a focus on detection, leaving corrective activities for future research. In some ways, our system is similar to the systems described in [11] and [12] which employ "governors" and "coordination artifacts" respectively, but these works are primarily focussed on helping the MAS work with external agents, and not necessarily on detecting when those agents violate social norms.

While some work has been done in detecting broken social commitments, it was done under the assumption that the MAS is aware of all major events [8]. These events are given as logical statements. Work has not been done on how a non-logical system would map messages and perceived activities to these logical statements. Here we will attempt to define the requirements that must be met for the system to be aware of all acts that break social commitments. Based on these requirements, we have created a system to detect broken social commitments.

Section 2 outlines the social commitment model that will be used throughout the remainder of this paper. Section 3 breifly describes CASA (Cooperative Agent System Architechture), the basic MAS that was expanded to allow for a social commitment observer. Section 4 details the requirements that must be met in order to detect broken social commitments. Section 5 is a discussion of the details of the implementation of a social commitment observer in CASA. Section 6 offers a conclusion and suggests directions for further research.

2 Social Commitments

Before a system can be designed to detect broken social commitments, social commitments have to be defined and the procedure for creating and disolving commitments must be outlined. We will draw on the model of social commitments outlined in [6] and [8]. This model has been chosen because Alberti and others have already shown that commitments can be detected using various forms of logic. This paper shows how we have implemented the ability to detect broken commitments in CASA.

A commitment is defined as a set including a debtor (x), a creditor (y), a condition (p) and a context (G) [6]. Together, the commitment states that x is committed to y to ensure that p comes about within some social context G. For the remainder of this paper, G will be assumed to be the system outlined in this paper, and is therefore the same in all of our cases. In addition, we informally add to all social commitments a timout (t), which gives the time that a commitment must be fulfilled by. Formally, this is part of the condition, in the form, "p will be fulfilled on or before time t," but for ease of discussion, it will be listed as a separate field in this paper.

Social commitments are formed, modified, and removed using one of the following actions [6]:

create. This action creates a commitment. In our system, this can result from any of the policies, and usually an agent becomes the debtor only when it sends or is sent a message.

discharge. This action occurs when a commitment's condition has been met, and therefore fulfills the commitment. This requires no action by the debtor or creditor other than those actions necessary to bring about the condition. Our system considers this a resolution that meets our social norms (a "good" resolution).

cancel. This action removes a commitment from a debtor, without the consent of the creditor. This is essentially a statement saying that an agent does not intend to fulfill its commitment, and will probably break it. In our system, however, a commitment is not technically broken (and therefore a social norm is not violated) until either an action occurs that makes the condition impossible to fulfill or the timeout is reached without the condition being fulfilled.

release. This action removes a commitment from a debtor with the permission of the creditor. This is considered acceptable within the social norms of our system.

delegate. This action changes the debtor field of the commitment. It requires the permission of the new debtor. Essentially, we are stating that if an agent commits to perform an action that another agent was committed to performing, that agent has passed the responsibility and is no longer required to bring about the condition. However, it could be argued that an agent is still committed, and would be at fault if the other agent did not fulfill the commitment.

assign. This action changes the creditor field of the commitment. It requires the permission of the old creditor.

The social context, G, determines when each of the actions can be performed. We have informally described when these actions are applied in our system, but the details of these *conversation policies* are described in the next section.

It is important to note that although social commitments define acceptable behaviors in our system, agents do not have to be internally aware of social commitments. In other words, when programming an agent, the programmer need not focus on social commitments so long as the agent will, in the end, act in accordance with the policies and the social commitments they create.

2.1 Conversation Policies

Conversation policies are rules that indicate when actions can and should be performed on social commitments. Our system adopts the conversation policies informally described in Table 1. These policies outline acceptable behaviors in and form the basis of our system. The P-propose policy indicates that a certain

Table 1. An informal description of the conversation policies as defined by Flores and Kremer [13]

Policy	Description
P-propose	A proposal commits the proposed agents to reply.
P-counter-offer	A counter-offer is considered a reply, and commits addressees to reply.
P-reply-acc	An acceptance releases proposed agents from the commitment to reply and releases counter-offered agents from the commitment to reply.
P-reply-rej	A rejection releases proposed agents from the commitment to reply and releases counter-offered agents from the commitment to reply.
P-reply-counter	A counter-offer releases proposed agents from the commitment to reply and releases counter-offered agents from the commitment to reply.
P-accept	An acceptance causes the formation of the proposed/counter-offered commitment.
P-release	A release releases the debtor of the given commitment, if sent by the creditor.

Table 2. An informal description of conversation policies based on the fish auction policies described by Venkatraman and Singh [7]

Policy	Description
FA-advertise	An advertisement at some price commits the advertiser to sending fish to the bidder if there is one and only one bid within a given time.
FA-bid	A bid commits the bidder to sending money if it receives fish from the advertiser.
FA-bad	A bad fish message essentially cancels the process, and therefore removes both the advertiser and bidder's commitments in relation to that fish.

degree of politeness is required of agents in the system. The requirements could be amended (politeness does not have to be a requirement) if a system designer desires a more open system.

It is possible for system designers to add new policies to their system. This allows other domain specific policies to be put into place when they would aid in understanding the expectations of agents participating in that system. For example, we have implemented a set of policies that define a fish auction [7]. These are informally described in Table 2. While there are other possible messages in the fish auction, they can all be inferred from these policies. Because the basic set of policies in our system include a way to set up arbitrary commitments (with the P-propose and P-accept policies, among others), new policies do not necessarily need to be created to use other protocols with this system.

3 CASA (Cooperative Agent System Architecture)

The work described in this paper expands upon CASA, a communication-based multi-agent system written in Java. A few of CASA's unique features are used to aid in the development, but any flexible MAS could be used as a basis for this work, with some modifications.

Figure 1 shows a typical run-time configuration of CASA. Every machine running CASA agents runs a special agent called the *local area coordinator* (LAC). The LAC is responsible for resolving agent addresses, keeping track of how to start up agents, and starting agents on behalf of other agents (that may be running on other machines). The CASA framework demands very little of agents running within it, but agents are expected to register with the LAC on start up, and may register information about how they can be re-started (if they want to offer services to other agents on demand). The message contents are standardized to a superset of the FIPA message standard [14].[1] Once registered, CASA agents are free to communicate with one another using the CASA message format over TCP/IP ports. CASA agents may also communicate through a special kind of agent called a cooperation domain, the subject of the next subsection.

[1] The actual messages can be in either XML [15] or a KQML-like [16] format.

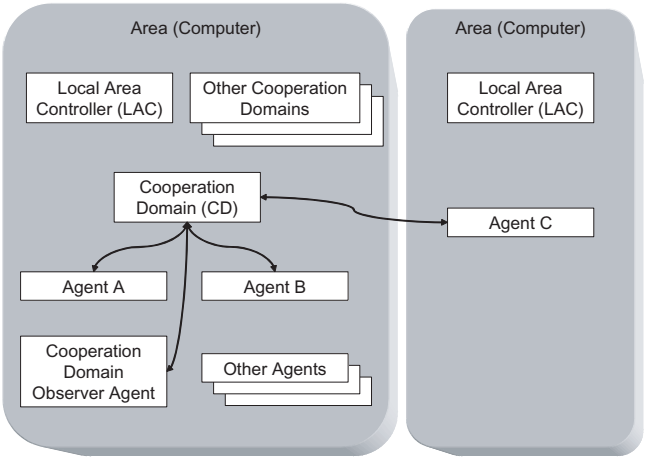

Fig. 1. A typical CASA run-time configuration

3.1 Cooperation Domains in CASA

A *cooperation domain* (CD) is an agent designed to aid agents in communicating in large groups. The CD allows agents to communicate with one another without knowing about every other agent. Agents register with the CD, and as a result they receive all non-private communications that are sent to the CD (including those *they* send). In this paper, a "cooperation domain" refers to either an agent itself or to a virtual location within which all agents (registered to that CD) operate. Figure 1 shows four agents participating in a conversation through a cooperation domain, depicted by the double-headed arrows.[2]

CASA agents are free to communicate directly (not through the CD) but they lose the power of the services potentially offered through the CD. Another advantage of the CD is that it gives the creators of the CD the ability to monitor the communications between its members. Figure 1 shows one such agent, the "Cooperation Domain Observer Agent," performing a special role within an agent conversation. This type of privileged agent can "eavesdrop" on all messages going through the cooperation domain, and is necessary when implementing a social commitment observer (see Sect. 5).

4 Detection Conditions

In order for the social commitment observer agent (or just *observer*) to detect broken social commitments, certain requirements must be met. The detection of broken social commitments is inferred from the observer's observations, the

[2] Messages sent in this way may be directed to all the participants (*broadcast*), to a specific subset of the participants (*multi-cast*), to a single agent (*directed*), or to all participants who have a particular *role* in the conversation (*role-cast*).

Table 3. Requirements for a social commitment observer to detect that a social commitment was formed

Requirement	Description
R1-understand	The observer understands social commitments and their structure.
R1-form	The observer observed the act that formed the social commitment. This may be either R1-form-accept or R1-form-policy.
R1-form-accept	The observer observed the acceptance of the social commitment (P-accept).
R1-form-policy	The observer observed an act that, because of a known conversation policy, automatically forms a social commitment (P-propose, for example).

way Sherlock Holmes solves a crime by decoding clues. This means that the observer does not rely on complaints or other error messages to determine if commitments are broken. The observer must infer that a social commitment is broken by observing the communications within the system and using any other means of apprehension it may possess (such as the ability to perceive some environment).

To detect the formation of a social commitment, the following conditions must be met. The observer must understand social commitments (both the concept and their structure). The observer must also observe the action that causes the formation of the commitment. This can happen in two ways. First, a social commitment is formed when an explicit request to form a social commitment was accepted by another agent, as defined by P-accept (see Table 1). A social commitment can also be formed through a conversation policy which results from some communication between agents. This can happen because of P-propose, P-counter-offer, P-inform, or any other policy that forms a commitment. This last requirement does not specify a particular set of policies because further policies can be added to a system by its developer (as mentioned in Sect. 2.1). Table 3 summarizes the conditions outlined above.

Once the observer agent has detected a social commitment it must store this commitment, as the commitment may be formed long before it is broken. The observer must then understand the condition portion of the social commitment.

Table 4. Requirements for a social commitment observer to detect that a social commitment was broken

Requirement	Description
R2-form	The observer detected that a commitment was formed.
R2-store	The observer has stored the commitment that was detected in R2-form.
R2-condition	The observer understands the condition part of the commitment.
R2-no-release	The observer has not observed an action that releases the debtor from the commitment.
R2-break	The observer observes an action that implies that the condition portion of the social commitment can never be satisfied.

This requirement is non-trivial, as our version of an open system includes the possibility of agents not understanding all other commitments. In the case where a commitment has been dissolved properly through one of the conversation policies (described in Sect. 2.1), nothing further will be required of the debtor. Therefore, the observer must detect a commitment that has not been properly dissolved. This implies that the observer is observing all activity of the agent, so that it isn't possible that the observer has missed the proper dissolution of the commitment. Finally, some action must happen that implies that the condition portion of the social commitment can never be satisfied. This action must be observed by the observer. Table 4 summarizes the conditions outlined above.

5 Implementation

In CASA, we implemented an agent that can detect broken social commitments (our *social commitment observer*). In doing so, we attempted to meet all of the requirements outlined in Sect. 4. In the following subsections we discuss how and to what degree we were able to meet each of the requirements.

5.1 Understanding Social Commitments (R1-understand)

The observer's understanding of social commitments begins with the understanding of the debtor and creditor fields of a social commitment. The FIPA standards define a *sender* and *receiver* field within every message. The *sender* field is always the agent that is currently sending the message while the *receiver* field is always the agent currently receiving the message. In CASA, when a cooperation domain is used to forward messages, the *sender* field is always the sending CD (to meet FIPA standards). Since the CD isn't (usually) the agent originally sending the message, it was necessary to add another field to messages within CASA (which is acceptable by FIPA standards). This is the *from* field. It is defined as the original sender of the message. Therefore, within CASA the *from* and *receiver* fields determine the debtor and the creditor of a given commitment. The *from* and *receiver* fields of the message are always *URLDescriptors*, which are used within CASA to both uniquely define an agent and define how to communicate with it (locally or across a network).

5.2 Observing Formation of Social Commitments (R1-form)

To observe the creation of all social commitments, the observer ties into a cooperation domain as a cooperation domain observer (Section 3.1 briefly describes this functionality). This allows the observer to meet the R1-form requirement as described below.

Once each message is received by the social commitment observer, it is processed to determine which conversation policies apply and therefore which commitments must be added to the set of current commitments. This is done by applying each known policy, in turn to the given message. The policies are parsed

in no particular order, and the successful operation of one policy does not imply that the other policies will not apply to the message. The addition of a new policy into the system requires only the creation of a new ConversationPolicy subclass.

5.3 Storing Social Commitments (R2-store)

CommitmentEngine objects store social commitments in a map from (debtor× creditor) to a set of conditions. In other words, given a debtor and a creditor, the agent can retrieve a set that defines the conditions that the debtor is committed to bringing about for the creditor. The conditions need not be understood at the point of storage, and may be stored in some general format, such as a string or a bit vector.

5.4 Understanding Conditions (R2-condition)

For any agent, there is a condition that is not understood. This is because our system does not put a restriction on the language used in describing the required condition of a social commitment. Therefore, any finite system will not be able to understand all social commitments. With this difficulty in mind, we have decided to implement the observer such that it only understands the commitments described explicitly by one or more conversation policies. It can still parse that an agent has formed a social commitment because of P-accept, but it may not be able to parse the condition portion of that commitment. In this case, the observer cannot detect when that commitment has been broken. As described below in Sect. 5.5, the observer is still able to detect when a debtor and creditor agree that the debtor should be released from its commitment (with the P-release policy). Future work will focus on this restriction (see Sect. 6).

5.5 Observing the Release from Social Commitments (R2-no-release)

The conversation policies used in Sect. 5.2 are responsible for creating commitments as they are observed. In addition, these policies are responsible for removing commitments from the set of all commitments when they are properly dissolved. This is the case with the P-ack, P-reply, and P-release policies.

Because agents are free to communicate outside of a cooperation domain, it is possible for the following scenario to take place. A message is sent by an agent, Alice, within a cooperation domain that forms a social commitment, and that commitment is detected and stored by the social commitment observer. Then, Alice (or another agent) sends a message that should release Alice from that commitment, but the message is sent outside of the cooperation domain. In this case, the commitment may be marked as broken at some time, even though it was actually properly dissolved. Our system requires that if an agent performs an action that creates a social commitment within a cooperation domain, any message properly dissolving that commitment must also be sent within the

cooperation domain. This requirement is not enforceable within CASA. If the requirement is not met, the system may detect that the agent has broken a social commitment, and act as if the agent has broken a social commitment.

5.6 Observing Broken Social Commitments (R2-break)

Like the formation and proper dissolution of commitments, the social commitment observer only detects broken social commitments if the message (or messages) that breaks the commitment is transmitted through the cooperation domain. The main problem with this requirement is not the difficulty of observing the action that breaks a commitment, but the fact that for some commitments, there is no such action. For example, let us assume that an agent, Alice, has a social commitment to another agent, Bob, to send him a message. If we assume that Alice and Bob are computer programs, and will therefore last as long as they are needed, and that we don't care about events beyond the end of the universe (if the universe does end), then Alice will always be able to send Bob a message at some time in the future, and no action would prevent this from occurring.

Because some commitments are not breakable, we have added another field to social commitments: a timeout value. This follows naturally from the fact that there is a *timeout* field in every CASA message. This still fits within our formal definition, because it can be thought of as an addition to the condition portion of the commitment similar to, "This condition will be brought about before *timeout*." The timeout value can be set so that the social commitment never times out (if the designer wishes), but as long as it does, the commitment will eventually be either fulfilled or broken. With a timeout value, we can modify the above example so that Alice has a social commitment to Bob to send him a message before August 1, 2005. This commitment will obviously be broken if Alice has not sent a message to Bob by the specified date. The P-inform and P-request conversation policies outlined in Sect. 2 both have timeouts in the CASA system, and so are easily monitored for breakage. The timeouts of all commitments are checked within the commitment engine every time the *expireCommitments()* function is called. Any commitments broken by the passage of time are treated as if a policy had determined that the commitment has been broken.

5.7 Initial Results

With the above requirements generally fulfilled, our social commitment observer is able to detect agents that fail to reply to requests, don't acknowledge messages when requested, and those that fail to complete correctly the fish auction as defined in [7].

6 Conclusion / Future Work

In this paper we have presented an implementation of a working social commitment observer in an open system. Towards this end, we have outlined the

requirements for an observer to detect that a social commitment was formed, and the requirements for an observer to detect that a social commitment was broken. Finally, we gave a detailed description of how we met each of the requirements for detecting broken social commitments. While we feel that we have made good progress in this area, there are several directions for future research.

It would be advantageous to be able to dynamically add policies and known conditions to the set that the social commitment observer understands. This would allow agents to define new requirements for their domains, while maintaining a central authority on commitments within a given cooperation domain. It may be possible to use the act and performative lattices built into CASA to store commitment information to aid the social commitment observer. This is because these lattices can be expanded as needed for each agent, and can then be passed from one agent to another with a standardized request.

It may also be beneficial to add a standard way for agents to "complain" about other agents that have broken social commitments. Because the social commitment observer presented here can detect that any type of commitment has been formed, the observer could confirm that indeed there was a commitment formed between the two agents. There would then have to be a way to determine when complaints are legitimate. This would probably require both the agent that registers the complaint and the agent that is complained about to be aware of the complaint verification process. Any agent unaware of this process would be unfairly judged, because it couldn't aid in the verification process. A comparative analysis of the transparent observation system and a complaint system is a future direction for investigation.

Finally, the social commitment observer can detect when a commitment is broken, but currently only displays a message to the user or writes an entry in a log file. In a system that may involve many communications at any time of day, some form of automated "punishment" to be meted out to agents that have broken social commitments may be necessary. The simplest punishment would probably be the ejection of agents that have broken a specified number of social commitments (or a certain number of commitments per time period). This ejection could be temporary or permanant. An alternative to the ejection of undesirable agents is to provide a service similar to the Better Business Bureau found in many cities. This service would give the number and/or type of commitments broken by some agent at another agent's request.

References

1. Denzinger, J.: Knowledge-based distributed search using teamwork. In: Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, USA (1995) 81–88
2. Hewitt, C.E.: The challenge of open systems. Byte **10** (1985) 223–242
3. Jennings, N.R., Campos, J.R.: Towards a social level characterisation of socially responsible agents. IEEE Proceedings on Software Engineering **144** (1997) 11–25
4. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, USA (1995) 41–48

5. Singh, M.: Social and psychological commitments in multiagent systems. In: AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels, Monterey, California (1991)
6. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law **7** (1999) 97–113
7. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols. Autonomous Agents and Multi-Agent Systems **2** (1999) 217–236
8. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interaction using social integrity constraints. In: Proceedings of the First International Workshop on Logic and Communication in Multi-Agent Systems (LCMAS 2003). (2003)
9. Pasquier, P., Flores, R., Chaib-draa, B.: Modelling flexible social commitments and their enforcement. In: Proceedings of the Fifth International Workshop on Engineering Societies in the Agents World (ESAW04). (2004)
10. Klein, M., Dellarocas, C.: Domain-independent exception handling services that increase robustness in open multi-agent systems. Working Paper ASES-WP-2000-02, Center for Coordination Science, Massachusetts Institute of Technology, Cambridge, MA, USA (2000) http://ccs.mit.edu/ases.
11. Esteva, M., Padget, J.A., Sierra, C.: Formalizing a language for institutions and norms. In: ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII, Springer-Verlag (2002) 348–366
12. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In: Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS04). Volume 1. (2004) 286–293
13. Flores, R., Kremer, R.: To commit or not to commit: Modelling agent conversations for action. Computational Intelligence **18** (2003) 120–173
14. Foundation for Intelligent Physical Agents (FIPA): FIPA ACL message structure specification. document number SC00061G, FIPA TC communication. (2003) http://www.fipa.org/specs/fipa00061/SC00061G.html.
15. World Wide Web Consortium (W3C): Extensible markup language (XML) (2004) http://www.w3.org/XML/.
16. Finin, T., Labrou, Y., Mayfield, J.: KQML as an agent communication language. In Bradshaw, J., ed.: Software Agents, MIT Press (1997) 291–316