A Formal Semantics for Gaia Liveness Rules and Expressions

Tim Miller and Peter McBurney[*]
Department of Computer Science,
University of Liverpool, Liverpool, L69 7ZF, UK
{tim, p.j.mcburney}@csc.liv.ac.uk

Abstract

The Gaia methodology is a development methodology for multi-agent systems that uses the concept of *roles* to define behaviour. Gaia uses *liveness expressions*, which are expressions written in a formal syntax that are used to define the ongoing behaviour of a role; and *liveness rules*, which are expressions specifying the behaviour of roles relative to each other in a system. However, while the syntax is formal, a formal semantics has not been defined, and there is no theory for how to reason about and manipulate these expressions. In this paper, we present a formal semantics for liveness rules and expressions, and discuss our work in developing axioms about them. We also discuss the introduction of a new operator for defining the *complement* of expressions; that is, the behaviour that falls outside of the liveness expression. This provides more flexibility when reasoning about and manipulating these expressions.

Keywords: multi-agent systems, agent-oriented development methodologies, Gaia design methodology, liveness expressions, Kleene algebra.

Biographies

Dr Peter McBurney is a senior lecturer in the Department of Computer Science at the University of Liverpool, UK, where he undertakes research on multi-agent systems, agent communications languages and computational economics. From 2004-2006, he was the project Co-ordinator for AgentLink III, a research co-ordination action project funded by the EC to promote European R&D in agent-based computing. He recently became co-editor of the journal, "The Knowledge Engineering Review".

Tim Miller received his PhD from the University of Queensland, Australia in 2005. He is now a research associate in the Agent ART group at the University of Liverpool, UK. His research interests include formal software development, multi-agent systems, agent interaction, and software testing. He is an active participant in the Community Z Tools project.

1 Introduction

The Gaia methodology [20] is a development methodology for multi-agent systems that borrows concepts and notations from the object-oriented development methodology, FUSION [3]. One such

[*]The authors are supported by the *Personalized Information Platform for Health and Life Sciences (PIPS)* Project (EC-FP6-IST-507019) and from the UK EPSRC *Market Based Control of Complex Computational Systems* Project (GR/T10657/01), respectively.

concept is *liveness expressions*, which are expressions that resemble regular expressions, and are used to specify the ongoing behaviour of an agent playing a role. Rules that quantify over these expressions, called *liveness rules*, specify the behaviour of roles relative to other roles in the system. Gaia roles are explained in detail in Section 2.2.

While the syntax of these expressions is formal, a formal semantics has not been defined, and there is no theory for how to reason about and manipulate these expressions. This is largely in part because the designers of Gaia have aimed to keep Gaia at an informal level. However, in an ongoing project in which we are using Gaia, we have found that, while Gaia is suitable to our needs for analysis, later in the development lifecycle we benefit from more formal languages. Instead of assessing and choosing a language that suited our requirements, we believe that providing a formal semantics that is agreed between project participants would increase the flexibility and usefulness of Gaia. Our formalism allows liveness expressions to better guide the design and development of agents without having to commit to a particular development language, while at the same time, it maintains the usefulness of Gaia in earlier stages of development by preserving the existing syntax.

In this paper, we present a formal semantics for liveness rules and expressions, which is based around traces of events, similar to that of CSP [5]. An event is either a protocol or action that an agent playing the role can participate in or perform, and is explicitly specified by a role specification. Each liveness expression defines a set of traces of events, and the trace of events in which an agent participates must be one of the traces in that set. Each liveness rule further specifies the behaviour of a role relative to the other roles in the system. The semantics do not change the existing syntax, because these are purposefully designed to be straightforward for developers to use, even if they have little background in the area.

Our formalism of liveness expressions is proven to satisfy the properties of a *Kleene algebra* [8], the formalism of regular expressions. This allows us to take advantage of a large body of work on Kleene algebra, such as the axiomatisation of Kleene algebras given by Kozen [9], which provides a sound basis for tool support, such as theorem provers and standard programming tools for regular expression matching.

We also introduce a new operator for defining the *complement* of expressions; that is, the set of traces that are the complement of the traces specified by a liveness expression. This provides more flexibility when reasoning about and manipulating these expressions. An axiom system for this complement operator is proposed, and proved to be sound and complete. This axiom system allows one to reason about and manipulate liveness rules and expressions, and specify desired or undesired system properties.

The work presented in this paper has been used on the European Commission-funded *Personalised Information Platform for Life & Health Services* (PIPS) project. The PIPS system is aimed to promote compliance with personalised medical advice from nutritionists and health-care professionals, as well as to provide personalised advice on best practices for prevention. The Gaia methodology has been used for analysis and specification of part of a *decision support system*, and the role specifications, complete with formalised liveness expressions, were used to guide the design and development of some agents in the system, as well as to identify exceptional behaviour. Details of project and of the decision support system can be found at the PIPS website[1], and the specification and design of the agent system can be found in [16].

This paper is outlined as follows. In Section 2, we discuss related work and give an overview of the relevant parts of the Gaia methodology. In Section 3, we define a formal semantics of Gaia liveness expressions. In Section 4, we present an additional operator for defining complement behaviour, and present an example of when this can be useful. Section 5 defines a formal semantics for Gaia liveness rules, and relates their semantics to liveness expressions. Section 6 concludes the paper.

[1] `http://www.pips.eu.org/`.

2 Background

In this section, we present some of the work most closely related to ours, and also give an overview of the relevant parts of the Gaia development methodology.

2.1 Related Work

Our semantics for Gaia liveness expressions and rules resemble that of process algebras. For example, we model the ongoing behaviour of a role and the interaction between roles as traces of events that occur with a particular set of orderings. Process algebras model the ongoing behaviour of processes and their interactions in a similar way. While there are several process algebras that model processes and their interactions, such as the π-calculus [13] and CCS [12], our formalism is most closely related to that of CSP [5]. A specification of a process in CSP contains the set of events in which the process can take part, called the *alphabet* of the process, and the set of possible event traces that the process can perform, including communication between other processes. A specification of a system in CSP is the combination of those processes, which can be either synchronous or asynchronous. While our semantics do not allow communication to be explicitly modelled, our semantics are event-based and a system specification is modelled as a combination of the traces defined by a set of roles.

Our formalism for Gaia liveness expressions is proved to be a Kleene algebra [9], which is a set of axioms derived for regular expressions [4]. This allows us to take advantage of any axioms derived for Kleene algebras, such as those in [8] and [9], use theorem provers for Kleene algebras, such as KAT-ML [1], an interactive theorem prover for an extension of Kleene algebra called *Kleene algebra with tests* [10], and use standard tools for regular expressions, such as regular expression matching tools in many programming languages.

Several agent-oriented development methodologies exist that serve similar purposes to Gaia. The partners in our project made a choice to use Gaia for development because of it is straightforward for people to learn, and because several key development people were familiar with it already.

The AAII methodology [7] was developed at the Australian Artificial Intelligence Institute (AAII) drawing on experience from several major agent applications that the AAII had developed with industry partners. The model that results from using AAII is closely related to the Distributed Multi-Agent Reasoning System (dMARS) [17], also developed at the AAII. The AAII methodology is based on object-oriented technology and ideas, and is extended with agent-specific concepts, including *roles*, a concept found in Gaia. However, the AAII methodology proposes no way to formalise role specifications.

ROADMAP [6] (Role-Oriented Analysis and Design for Multi-Agent Programming) is an extension of the Gaia methodology, designed to address what the ROADMAP authors saw as problems with the initial versions of Gaia, some of which were addressed in later versions of Gaia itself. Most of these extensions are at the analysis level, such as adding a requirements gathering phase. In addition, roles are modelled using hierarchies in order to provide different levels of abstraction during the analysis phase. As with Gaia, role specifications are not formal.

The MaSE [19] (Multi-agent Systems Engineering) methodology is agent-oriented methodology that uses *role models* to document roles. Role models specify which of the system goals a role is responsible to maintain or achieve. The goals are structured using a goal hierarchy, with the more important goals at the top of the hierarchy. The authors of [19] present only informally specified goals, while the ongoing behaviour of agents is derived from these goals, rather than explicit representations such as liveness expressions.

Formal Tropos [15], an integration of formal methods and the Tropos agent-oriented software engineering methodology [2], is one development methodology that does provide a way for modelling role behaviour formally. However, the language is based on temporal logic, which is far more rigorous and heavy than the straightforward notation used in Gaia.

Prometheus [14] is a methodology that draws ideas from several different areas, but is especially based on the JACK agent platform[2]. Prometheus does not cover the entire development process — no implementation phase is specified. However, the detailed design phase is clearly aimed at developing agents for the JACK platform, and as a result, Prometheus is not as widely applicable as a methodology such as Gaia.

The formalism in this paper is not directly applicable to any of the methodologies discussed in this section because none of these use liveness expressions, other than ROADMAP, which is based on Gaia. However, with the exception of Formal Tropos, which is already complete in terms of formality, we believe that it would be beneficial to integrate the role liveness expressions into other languages. Liveness expressions could be integrated into the AAII and Prometheus methodologies in a straightforward manner, because they offer no other way to formally specify role behaviour. Liveness expressions could be used to explicitly specify role behaviour in MaSE; however, the atomic events would have to be tied to the specified goals for this to be useful, so the goals and liveness expressions would have to be managed together.

2.2 Gaia

The Gaia methodology [20] is a development methodology for multi-agent systems that borrows concepts and notations from the object-oriented development methodology, FUSION [3].

Gaia uses the metaphor of a "human organisation" to identify the roles and environment of the system being developed. While other development methodologies also use roles, Gaia's use of this metaphor gives developers a way to structure their development using a familiar concept. Thus, the view of a system analysed and designed using Gaia is an organisation in which agents play one or more roles. This is similar to a human organisation in which one person may fulfil many roles in a company, e.g. programmer and code reviewer. The relationship between roles and agents can be dynamic, in that an agent can fulfil different roles, and a role can be fulfilled by many agents throughout the lifetime of the system.

Roles in Gaia consist of four key attributes:

- *Responsibilities*: the responsibilities of the role in the context of the organisation, expressed as properties. These are divided into two categories: *liveness* properties, which are the goals that the agent should achieve in the system; and *safety* properties, which are invariants that the agent must maintain throughout the evolution of the system. These determine the functionality of an agents.
- *Permissions*: the rights that a role has within the system to access certain information, or perform certain actions on the environment or other agents.
- *Activities*: the actions that can be performed by a role without the interaction of other agents. These can be viewed as private actions.
- *Protocols*: the protocols of a role define the way that a role can interact with other roles in the system.

[2]See `http://www.agent-software.com.au/`.

Safety properties are quantified over the environment of the system, and therefore a number of formal languages, such as Z [18], could be used to express these properties.

Liveness properties, on the other hand, do not have a formal definition, and the syntax is set by Gaia. Liveness properties are specified using *liveness expressions*, which borrow their syntax and semantics from FUSION [3]. The syntax is based on regular expressions, with the idea that this gives a straightforward notation with which many developers are already familiar. Operators for this notation are outlined in Table 1. Unary operators have the highest precedence, followed, in order from highest to lowest, by '.', '$[]$', and finally, '$\|$'.

Operator	Informal definition
x.y	x occurs followed by y
$x \mathbin{[]} y$	x occurs or y occurs
x^*	x occurs zero or more times
x^+	x occurs one or more times
x^k	x occurs exactly k times
[x]	x occurs optionally
$x \parallel y$	x and y occur concurrently

Table 1: Operators for Liveness Expressions

If a represents atomic actions, then the grammar for liveness expressions is as follows:

$$\Omega \quad := \quad a \quad | \quad \Omega \,.\, \Omega \quad | \quad \Omega \mathbin{[]} \Omega \quad | \quad \Omega^* \quad | \quad \Omega^+ \quad | \quad \Omega^k \quad | \quad [\,\Omega\,] \quad | \quad \Omega \parallel \Omega \quad |$$

Liveness expression *definitions* take the form:

$$\textsc{Role} = \textit{Expression}$$

in which Role is the name of a role, and *Expression* defines a liveness expression that adheres to the above grammar.

As an example, we used the conference review system from Zambonelli *et al.* [21]. This example models a process and set of roles for handling the review process of a conference. This process is broken into phases: submission, review, and the publication of accepted paper. In the review phase, the role of *Reviewer* is to review a paper and send back their review form. In [21], this role is modelled using the following liveness expression:

$$\textsc{Reviewer} = (\mathsf{ReceivePaper.ReviewPaper.SendReviewForm})^{maximum\text{-}number}$$

This liveness expression specifies a process that first the reviewer receives the paper, then reviews the paper, and then send the review form back. It does this *maximum-number* times, in which *maximum-number* is the maximum number of papers that a reviewer is permitted to review.

The atomic components in a liveness expression, such as ReceivePaper and ReviewPaper, are either *activities*, which are the atomic units of action that a role can perform without interacting with other agents, or *protocols*, which are the aspects of behaviour that require interaction with other agents. We use the term *events* to refer to both activities and protocols, and they are treated as the same concept in the formalism. All protocols and activities in which a role takes part are explicitly specified in the role specification. We call this set of activities the *alphabet* of the role.

Organisational rules are the rules that specify how roles in a system behaviour relative to each other. For example, if we have two roles, R and Q, we can specify a liveness rule that states that role R must be played three times before role Q can be played: $R^3 \to Q$. Similar to the responsibilities in roles, organisational rules can be divided into liveness rules and safety rules.

The operators for liveness rules notation are outlined in Table 2[3]. These operators are similar to those of liveness expressions, except that they allow the referencing of roles that have been previously declared.

Operator	Meaning
$L \to M$	L must occur before M
$L \vee M$	either L or M must occur, but not both
$L \wedge M$	both L and M must occur
L^*	L must occur zero or more times
L^+	L must occur one or more times
L^k	L must occur k times
$[L]$	L occurs optionally
$L \parallel M$	L and M occur concurrently
$\neg L$	L does not occur
$L^{k..m}$	L occurs between k and m times

Table 2: Operators for Liveness Rules

3 Formalising Liveness Expressions

While the syntax for Gaia liveness expressions is formal, the semantics is defined informally. This implies that different developers can interpret liveness expressions in difference ways, and only simple reasoning can be performed over an expression. Additionally, axioms for for rewriting liveness expressions are difficult to define without a formal semantics. In this section, we define a denotational semantics for liveness expressions.

We do not believe that the informal semantics specified in Table 1 is sufficient. While the definitions are straightforward, they are also ambiguous. For example, the expression $a.b.c \parallel d.e.f$, in which each letter is atomic, could be interpreted in several different ways, such as:

- Either $a.b.c$ executes and then $d.e.f$ executes, or vice-versa.
- a and d execute, followed by b and e, then c and f.
- Atomic actions a and d (and similar for the rest) are executed at the same time.
- a, b, c, d, e, f execute in any order, provided that a executes before b, and b before c, and similarly, d executes before e, and e before f.

The last of these interpretations is a more general case than the first two, in that the set of traces for this interpretation is a superset of the traces from the first two. The third interpretation is different altogether, in that it allows atomic events to occur simultaneously. Without defining the

[3]Some of these operators are not explicitly defined in [20] or other Gaia literature, but we have extrapolated them from examples and from the operators in Table 1.

semantics of this operator, documents containing such expressions can be interpreted differently by different readers.

The semantics presented in this paper would define the behaviour as being the last of the above alternatives, and we define events such that they are atomic and never occur simultaneously. More specifically, we mean that atomic events are considered executed only once the execution is complete, and the completion of two such events cannot be observed at the same time.

The semantics is similar to that of CSP [5], in which a process defines a set of traces of events. In our semantics, an event is an action or a protocol. Take for example, the simple expression $a.(b \mathbin{[\,]} c).d$. This representes the traces $\langle a, b, d\rangle$ and $\langle a, c, d\rangle$, so the set of all traces is a set containing precisely those two traces. The trace of an agent fulfilling a role must be one of the traces in the set of traces specified by the role's liveness expression.

Throughout this paper, we use Σ to represent the set of all events used by a role, a and b as variables representing elements in Σ, Σ^* as the set of all sequences over Σ, s, t, u, and v as variables representing elements in Σ^*, Ω as the set of all liveness expressions, and x, y, and z as variables representing elements in Ω; that is, composites of atomic events. The set of traces defined by a liveness expression x is written $[\![x]\!]^{\mathcal{T}}$, and we note that $[\![x]\!]^{\mathcal{T}} \in \mathcal{P}(\Sigma^*)$ for all x, in which $\mathcal{P}$ is the power set function.

3.1 Constants and Literals

To help with the definition of our semantics, we introduce two constants into the liveness expression syntax:

- $\emptyset$: this defines the empty set of traces.
- ϵ: this defines the empty liveness expression. That is, the only behaviour is an empty sequence of events. This is distinguished from $\emptyset$ by the fact that $\emptyset$ defines no behaviour, whereas ϵ defines the behaviour consisting of no events.

Formally, these two constants are defined as follows:

$$\begin{array}{lcl} [\![\emptyset]\!]^{\mathcal{T}} & \widehat{=} & \emptyset \\ [\![\epsilon]\!]^{\mathcal{T}} & \widehat{=} & \{\langle\rangle\} \end{array}$$

However, ϵ can be defined in terms of $\emptyset$ and the * operator, whose semantics is defined in Section 3.2, as follows:

$$[\![\epsilon]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![\emptyset^*]\!]^{\mathcal{T}}$$

Recall that x^* is zero or more iterations of x. Therefore, we have that x^* is equivalent to $\epsilon \mathbin{[\,]} x \mathbin{[\,]} x.x \mathbin{[\,]} x.x.x \mathbin{[\,]} \ldots$, so $\emptyset^*$ is equivalent to $\epsilon \mathbin{[\,]} \emptyset \mathbin{[\,]} \emptyset.\emptyset \mathbin{[\,]} \emptyset.\emptyset.\emptyset \mathbin{[\,]} \ldots$. However, as proved in Appendix A, for any x, $x.\emptyset = \emptyset$ and $x \mathbin{[\,]} \emptyset = x$, therefore $\emptyset^*$ is equivalent to ϵ.

A *literal* is an atomic event. That is, a literal is a member of the set Σ, for example, ReviewPaper in the conference management example. The traces represented by a reference to a literal reference is a singleton set containing containing a trace with only one element: the literal. Formally, this is defined as follows:

$$[\![a]\!]^{\mathcal{T}} \quad \widehat{=} \quad \{\langle a\rangle\} \qquad \textbf{where } a \in \Sigma$$

Therefore, the set of traces from the atomic event ReviewPaper would be $\{\langle \mathsf{ReviewPaper}\rangle\}$.

3.2 Operators

There are seven operators defined in Gaia. Of these, we define four to be *primitive*. By primitive, we mean that their semantics are not defined solely by using other operators.

The first primitive definition is that of the '.' operator, called the *sequence* operator, which is defined as x occurring before y. Formally, we define the set of traces in $x.y$ as the concatenation of all traces in x with all traces in y:

$$[\![x.y]\!]^{\mathcal{T}} \quad \widehat{=} \quad \{s ^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}\}$$

As an example, we use part of the REVIEWER role from the conference management system. The act of reviewing one paper is specified using the expression $\mathsf{ReceivePaper.ReviewPaper.SendReviewForm}$. In this expression, $x = \mathsf{ReceivePaper}$ and $y = \mathsf{ReviewPaper.SendReviewForm}$. The set of traces from this sequential composition is $\{\langle \mathsf{ReceivePaper}, \mathsf{ReviewPaper}, \mathsf{SendReviewForm}\rangle\}$.

The next primitive definition is that of the '$[\,]$' operator, called the *choice* operator, which is defined as either x occurring or y occurring. Formally, we define the set of traces in $x \; [\,] \; y$ as the union of all traces in x with all traces in y:

$$[\![x \; [\,] \; y]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![x]\!]^{\mathcal{T}} \cup [\![y]\!]^{\mathcal{T}}$$

Consider that a reviewer may wish to decline that he/she reviews a paper. We can specify that the review process for a single paper is as follows:

$$\mathsf{ReceivePaper.(DeclineReview} \; [\,] \; \mathsf{ReviewPaper.SendReviewForm)}$$

Therefore, after receiving the paper, the reviewing can decline to review it, or can review it and then send the review form. The behaviour prescribed by $[\,]$ is the union of the traces on either side. $\mathsf{DeclineReview}$ is an atomic concept, and $\mathsf{ReviewPaper.SendReviewForm}$ is the sequential composition of two atomics, so their traces are $\{\langle \mathsf{DeclineReview}\rangle\}$ and $\{\langle \mathsf{ReviewPaper}, \mathsf{SendReviewForm}\rangle\}$, so the union of them is $\{\langle \mathsf{DeclineReview}\rangle, \langle \mathsf{ReviewPaper}, \mathsf{SendReviewForm}\rangle\}$.

The third primitive definition is that of the '*' operator, called the *iteration* operator, which is defined as x occurring 0 or more times iteratively. Formally, we define the set of traces in x^* as the distributed union of all traces in x^n (defined later in this section), for all $n \geq 0$:

$$[\![x^*]\!]^{\mathcal{T}} \quad \widehat{=} \quad \bigcup_{n \geq 0} [\![x^n]\!]^{\mathcal{T}}$$

Consider the case in which there is no maximum number of papers that a reviewer can review in the conference management system. One could represent this as follows:

$$(\mathsf{ReceivePaper.ReviewPaper.SendReviewForm})^*$$

This implies zero of more iterations of $\mathsf{ReceivePaper.ReviewPaper.SendReviewForm}$. Therefore, the set of traces is:

$$\begin{aligned}
&\{\emptyset,\\
&\langle \mathsf{ReceivePaper}, \mathsf{ReviewPaper}, \mathsf{SendReviewForm}\rangle,\\
&\langle \mathsf{ReceivePaper}, \mathsf{ReviewPaper}, \mathsf{SendReviewForm}, \mathsf{ReceivePaper}, \mathsf{ReviewPaper}, \mathsf{SendReviewForm}\rangle,\\
&\ldots\}
\end{aligned}$$

In which the ellipsis $(\ldots)$ represents the set of traces in which each trace is the same as the previous, but extended with an additional iteration. This gives us a countably infinite set of traces, the largest of which is countably infinite in length.

These three definitions are all quite trivial, and are common in definitions for regular expressions. However, the final primitive definition, that of the '$\|$' operator, called the *interleave* operator, is less straightforward. Formally, we define the set of all traces in $x \parallel y$ as the set of traces in which the events from x and y are interleaved. That is, for a liveness expression $x \parallel y$, any events from a trace from x or y must occur in the order expressed by that trace, but may be interleaved with events from the other. We use the recursive definition of the *interleaves* relation as defined for CSP [5], which states the following three laws:

$$
\begin{array}{ll}
\textbf{L1} & \langle\,\rangle \; interleaves(t,u) \Leftrightarrow t = u = \langle\,\rangle \\
\textbf{L2} & s \; interleaves(t,u) \Leftrightarrow s \; interleaves(u,t) \\
\textbf{L3} & \langle a \rangle ^\frown s \; interleaves(t,u) \Leftrightarrow \\
 & \qquad t \neq \langle\,\rangle \wedge t_0 = a \wedge s \; interleaves(t',u) \vee \\
 & \qquad u \neq \langle\,\rangle \wedge u_0 = a \wedge s \; interleaves(t,u')
\end{array}
$$

In these laws, we use the notation from [5] that for any sequence s, the head of the sequence is written as s_0, and the tail as s'.

Using this relation, we formally define the $\|$ operation as follows:

$$[\![x \parallel y]\!]^{\mathcal{T}} \quad \widehat{=} \quad \{s \mid (\exists t \in [\![x]\!]^{\mathcal{T}}, u \in [\![y]\!]^{\mathcal{T}} \mid s \; interleaves(t,u)\}$$

As an example, we consider an alternative conference management system in which reviewers must review exactly two papers, A and B. The reviewer will receive paper A before paper B, as this is determined by the program chair, but can review them and send the review forms back at any time afterwards, provided that the review is performed before the form is sent. This could be expressed as follows:

$$\mathsf{ReceivePaperA}.\mathsf{ReceivePaperB}.(\mathsf{ReviewPaperA}.\mathsf{SendReviewFormA} \parallel \mathsf{ReviewPaperB}.\mathsf{SendReviewFormB})$$

In which ReceivePaperA and ReceivePaperB represent the events of reviewing two different papers, and similarly for the other atomic events. The interleaving of the reviewing and form sending gives rise to a number of possible behaviours. A reviewer could review A, then send the form back for A, then do the same for B. Alternatively, a reviewer can review B, then review A, then send the form for A, finally sending the form for B. In total, there are six possible traces, which can be enumerated as follows, in which RA is ReviewPaperA, SA is SendReviewFormA, and similarly for paper B:

$$
\begin{array}{c}
\{\langle RA, SA, RB, SB\rangle, \langle RA, RB, SA, SB\rangle, \langle RA, RB, SB, SA\rangle, \\
\langle RB, SA, RA, SA\rangle, \langle RB, RA, SB, SA\rangle, \langle RB, RA, SA, SB\rangle\}
\end{array}
$$

In the technical report version of this paper [11], we provide and prove theorems about rewriting expressions such that all instances of the interleave operator can be removed from expressions and replaced with expressions using only the sequence, choice, and iteration operators, except expressions of the form $x^* \parallel y.z$ and $x^* \parallel y^*$, for which we do not have theorems. While we note that all instances of the interleave operator can be written using primitive operators, we do not believe universal theorems for doing so exist. This is an unfortunate property of the interleave operator, which is due to its inherent complexity, however, we do have rewrite rules for some specific instances of these expressions, such as the case in which $x \in \Sigma$. We also have refinement theorems about expressions of these forms, which give an approximation of a rewrite rule that is proved to be a subset of the traces in the original expression. While these rewrite theorems are not necessary to determine the semantics of an expression, they allow us to produce others theorems, such as those for the complement operator defined in Section 4, without having to consider the interleaving operator.

Now, we define the rest of the operators, which are all non-primitive. That is, they are defined solely in terms of the other operators.

First, we provide the semantics for x^k, called the *k-iteration* operator, which is defined as k iterations of x, for example, $x^3 = x.x.x$. Formally, we define the set of traces in x^k as the concatenation of x to itself k-1 times:

$$\begin{array}{lcll} [\![x^k]\!]^{\mathcal{T}} & \widehat{=} & [\![x.x^{k-1}]\!]^{\mathcal{T}} & \textbf{where } k > 0 \\ & \widehat{=} & [\![\epsilon]\!]^{\mathcal{T}} & \textbf{where } k \leq 0 \end{array}$$

The Reviewer role in the conference management system is an example of this. If we instantiate the parameter *maximum-number* with the value 2, then we can expand the expression using the definition to the following:

ReceivePaper.ReviewPaper.SendReviewForm.ReceivePaper.ReviewPaper.SendReviewForm

The single trace for this expression is straightforward to determine.

The next definition is that of the x^+ operator, called the *positive-iteration* operator, which is defined as x occurring one or more times iteratively. Formally, we define the set of traces in x^+ an occurrence of x, followed by 0 or more occurrences of x:

$$[\![x^+]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![x.x^*]\!]^{\mathcal{T}}$$

The final operator definition is that of the $[x]$ operator, called the *optional* operator, which is defined as x occurring once, or nothing occurring. Formally, we define this as the set of traces in x combined with the set of trace in ϵ:

$$[\![[x]]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![x \mathbin{[\!]} \epsilon]\!]^{\mathcal{T}}$$

Consider a case in which a reviewer may send an acknowledgement that he/she has received the paper, but this acknowledgement is optional. One could specify this as follows:

ReceivePaper.[SendAcknowledgement].ReviewPaper.SendReviewForm.

The set of traces corresponding to this expression is:

$\{\langle$ReceivePaper, ReviewPaper, SendReviewForm$\rangle$,
$\langle$ReceivePaper, SendAcknowledgement, ReviewPaper, SendReviewForm$\rangle\}$

3.3 Relationship to Kleene Algebras

Our formalism of liveness expressions forms a *Kleene algebra* [8]. A Kleene algebra is a commutative, idempotent semiring. That is, a Kleene algebra is defined as a structure $\mathcal{R}$ with binary operations '+' and '.', unary operator '*', and constants 0 and 1 such that $(\mathcal{R}, +, ., 0, 1)$ form a *idempotent semiring*. That is, the following properties hold:

1. $(\mathcal{R}, +)$ is a commutative monoid with identity 0. That is, for all x, y, and z in $\mathcal{R}$:
    - $(x + y) + z = x + (y + z)$ (+ is associative)
    - $x + 0 = x = 0 + x$ (0 is an identity for +)

- $x + y = y + x$ (+ is commutative)
- $x + x = x$ (+ is idempotent)

2. $(\mathcal{R}, .)$ is a monoid with identity 1. That is, for all x, y, and z in $\mathcal{R}$:
   - $(x.y).z = x.(y.z)$ (. is associative)
   - $1.x = x = x.1$ (1 is an identity for .)
3. . distributes over +. That is, for all x, y, and z in $\mathcal{R}$:
   - $x.(y + z) = x.y + x.z$ (. left-distributes over +)
   - $(x + y).z = x.z + y.z$ (. right-distributes over +)
4. 0 annihilates . on $\mathcal{R}$. That is, for all x in $\mathcal{R}$:
   - $x.0 = x = 0.x$

Additionally, the * operator satisfies the following properties for all x and y in $\mathcal{R}$:

- $1 + x.x^* \leq x^*$
- $1 + x^*.x \leq x^*$
- $x.y \leq y \Rightarrow x^*.y \leq y$
- $y.x \leq y \Rightarrow y.x^* \leq y$

in which the relation $\leq$ defines a partial order on $\mathcal{R}$, such that for all x, y in $\mathcal{R}$:

$x \leq y \Leftrightarrow x + y = y$

If we define liveness expressions as the tuple $(\Omega, [], ., \emptyset, \epsilon)$, in which Ω is defined as the set of all liveness expressions, then our formalism fulfils the properties of a Kleene algebra. This is proved in Appendix A.

Proving this allows us to take advantage of other work on Kleene algebras, such as the axiomatisation of Kleene algebras given by Kozen [9], and related tool support, such as KAT-ML [1], an interactive theorem prover for an extension of Kleene algebra called *Kleene algebra with tests*, and standard tools for regular expressions, such as regular expression matching tools in many programming languages.

In [11], we prove additional theorems about our liveness expressions, focusing on the theorems about the interleave operator, which is not part of Kleene algebra. Kleene algebra axioms, and the axioms defined by us in [11], allow us to reason about Gaia liveness expressions in a formal way, without changing the way in which such expressions are written and developed.

4 Complementary Behaviour

In this section, we define a *complement* operator for Gaia liveness expressions, which specifies all of the traces that are not in a liveness expression. The complement of a liveness expression, x, is written $\overline{x}$, and for all x, $[[\overline{x}]]^{\mathcal{T}} \in \mathcal{P}(\Sigma^*)$.

A complement operator is useful for reasoning about Gaia liveness expressions, and provides more flexibility in specifying roles. For example, the choice operator can be used as deterministic choice (or if-then-else), such that if x occurs, do y, otherwise, do z: $x.y \;[]\; \overline{x}.z$. While this is possible without the complement operator, the possible behaviours in $\overline{x}$ may not be straightforward to specify.

As an example, consider the conference management system from Zambonelli *et al.* [21]. Recall from earlier that the liveness expression for the Reviewer role is as follows.

$$\textsc{Reviewer} = (\mathsf{ReceivePaper}.\mathsf{ReviewPaper}.\mathsf{SendReviewForm})^{maximum\text{-}number}$$

After a certain time, the reviewer may have received a paper, but failed to perform the review or send the review form. In this case, the reviewer may receive a reminder from the program chair of their commitment to review the paper, denoted using the atomic expression ReceiveReminder. Using the complement operator, we can define this as follows.

$$\begin{aligned}\textsc{Reviewer} = (&\mathsf{ReceivePaper}.\mathsf{ReviewPaper}.\mathsf{SendReviewForm} \;[]\\ &\mathsf{ReceivePaper}.\overline{\mathsf{ReviewPaper}.\mathsf{SendReviewForm}}.\mathsf{ReceiveReminder})^{maximum\text{-}number}\end{aligned}$$

This expression says that, after the reviewer has received the paper, if the review is not completed and then returned, then the reviewer can expect to receive a reminder from the program chair. Using the complement operator is much more straightforward than specifying all the possible alternatives, for example, if the review is performed, but the form not sent back, if the review is not performed at all but an empty review form is sent, or neither.

Our motivation for the introduction of a complement operator is to help us identify *exceptional behaviour*. That is, all traces of events that are not valid behaviours as specified by a liveness expression. Specifically, we use the complement behaviour of a liveness expression to help with verification and exception handling. For example, test case selection techniques from formal specifications often advocate dividing the input space in disjoint equivalence classes, and selecting one test case from each class as a way to ensure certain levels of coverage. Similarly, disjoint equivalence classes of behaviour can be derived in order to determine the types of exceptions one must handle when designing an agent to interact with a certain role.

From this, the aims of the complement operator and its axioms can be clearly defined. We want to introduce them such that we can identify disjoint equivalence classes that document the behaviour of a role such that, if we have the traces, $\mathcal{R}$, for a liveness expression, and the equivalence classes $\mathcal{R}_1, \ldots, \mathcal{R}_n$ of behaviour that each define a choice between exceptional behaviour, we want the following properties to hold.

$$\forall i, j \in 1..n \mid i \neq j \Rightarrow \mathcal{R}_i \cap \mathcal{R}_j = \emptyset \tag{4.1}$$

$$\forall i \in 1..n \mid \mathcal{R} \cap \mathcal{R}_i = \emptyset \tag{4.2}$$

$$\mathcal{R} \cup \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n = \Sigma^* \tag{4.3}$$

That is, (4.1) each equivalence class of exceptional behaviour is disjoint from every other; (4.2) each equivalence class of exceptional behaviour is disjoint from the expected behaviour; and (4.3) the union of all behaviours is equivalent to Σ^*, the set of all possible behaviours.

4.1 A Complement Operator

First, we define what it means for a trace to be complementary to another. Formally, for a liveness expression, x, we define its complement, $\overline{x}$, as every trace in Σ^* that is not in x:

$$[\![\overline{x}]\!]^{\mathcal{T}} \quad \widehat{=} \quad \{s \mid s \notin [\![x]\!]^{\mathcal{T}}\}$$

In the conference management example, the complement of the REVIEWER role would be defined as any trace that is not *maximum-number* iterations of ReceivePaper, ReviewPaper, and SendReviewForm, such as receiving a new paper before completing the review of the first. The complement operator provides a compact notation for specifying this, rather than having to explicitly specify every case.

While this operator is enough to specify the complement of a liveness expression, it may not be useful if we want to reason about liveness expressions. Consider the REVIEWER example again. The complement of the liveness expression representing a review of one paper is as follows:

$$\overline{\mathsf{ReceivePaper.ReviewPaper.SendReviewForm}}$$

While this notation is compact, this expression may not be straightforward to reason about, and not straightforward to enumerate. For example, this can be violated by the agent playing this role sending a review form *before* receiving or reviewing the paper, or by receiving the paper and then failing to review it. For this reason, we propose an additional constant and an additional operator to Gaia liveness expressions.

4.1.1 Additional Constant

The additional constant that we introduce is $*$, called the *universal* constant, which defines the set of all possible traces in Σ. A primitive definition of this operator is as follows:

$$[\![*]\!]^{\mathcal{T}} \quad \widehat{=} \quad \Sigma^*$$

However, this can also be defined as a non-primitive constant by using the complement operator:

$$[\![*]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![\overline{\emptyset}]\!]^{\mathcal{T}}$$

That is, the universal constant is the complement of the empty set of traces. While $*$ is not any more of a shorthand then $\overline{\emptyset}$, we believe that the $*$ symbol is more intuitive. In general, we use $*$ in expressions of the form $x.*$, which defines the set of traces that have a trace from x as their prefix. For example, one of the complement behaviour of the REVIEWER role is if the reviewer first tries to perform any event other than receiving a paper. This can be specified using the expression $\overline{\mathsf{ReceivePaper}}.*$, which represents any behaviour that does not begin with receiving the paper.

This operator produces some interesting theorems, such as the following: $\epsilon \leq x \Rightarrow x.* = *$, which is proved in Appendix B, Theorem B.2(v).

4.1.2 Additional Operator

In addition to the complement operator, we introduce a binary operator, $\sqcap$, called the *conjunction* operator, for which $x \sqcap y$ defines the set of traces that are in both x and y. Formally, we define this as the intersection of the traces from x and y:

$$[\![x \sqcap y]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![x]\!]^{\mathcal{T}} \cap [\![y]\!]^{\mathcal{T}}$$

However, using the results of set theory, we know that the intersection of two sets, A and B, is the complement of the union of the complements of A and B. Therefore, we can define this as a non-primitive operator:

$$[\![x \sqcap y]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![\overline{\overline{x} \,[\!]\, \overline{y}}]\!]^{\mathcal{T}}$$

That is, $x \sqcap y$ is equivalent to the complement of the choice between $\overline{x}$ and $\overline{y}$. Clearly, the $\sqcap$ operator is more readable than its primitive equivalent, so we use this notation throughout the paper, however, defining this as shorthand is useful because it does not increase the complexity of the language.

This can be viewed as a conjunction, in that in order for a trace, t, to be in the set of traces for the expression $x \sqcap y$, t must be in both x and y. Therefore, if $[\![x]\!]^{\mathcal{T}}$ and $[\![y]\!]^{\mathcal{T}}$ are disjoint, $x \sqcap y$ is equivalent to $\emptyset$. For example, in the conference management system, Zambonelli *et al.* [21] discuss the Author role. If we want to specify the behaviour that is neither an Author nor a Reviewer, we can use $\overline{\text{Author}} \sqcap \overline{\text{Reviewer}}$ to specify the set of traces that are in both $\overline{\text{Author}}$ and $\overline{\text{Reviewer}}$.

This operator can also be used to specify behaviour comparable to set difference. That is, the expression $x \sqcap \overline{y}$ specifies that a role should behave as specified in x minus the behaviour in y. So, the set of traces $[\![x \sqcap \overline{y}]\!]^{\mathcal{T}}$ is equivalent to $[\![x]\!]^{\mathcal{T}} \setminus [\![y]\!]^{\mathcal{T}}$.

4.2 Normal Form

We propose a concept for liveness expressions called *normal form*. Normal form is analogous to the *disjunctive normal form* of Boolean logic. That is, a liveness expression is in normal form if and only if it is a choice between one or more conjunctions, iterations, sequential compositions, atomic expressions, complemented atomic expressions, complemented ϵ, or complemented $\emptyset$, each of which contain no choice operators. Additionally, sequential compositions must not contain any conjunctions. Simply put, writing a liveness expression consists of pushing all complement operators and sequential operators inwards, and pushing all choice operators outwards.

For example, the expression $\overline{a} \,[\!]\, (b \sqcap \overline{c})$, in which $a, b, c \in \Sigma$, is in normal form. However, the expression $(a \,[\!]\, b).\overline{c}$ is not in normal form, because the reference to the choice operator is not propagated outwards. Its equivalent, $a.\overline{c} \,[\!]\, b.\overline{c}$, is in normal form. The expression $\overline{a \,[\!]\, b}$ is not in normal form, because the complement operators is applied to a compound expression. Its equivalent, $\overline{a} \sqcap \overline{b}$ is in normal form.

There is one special case in which the complement of a compound expression is in normal form: the case in which an atomic event is an argument to the sequence operator, and $*$ is the other argument; for example $\overline{a.*}$. This represents any behaviour that does not start with a. Such an expression can not be reduced further. However, if a was a compound expression, further reduction would be possible.

4.3 Systematically Deriving the Normal Form of Liveness Expressions

We propose an axiom system that can be used to systematically derive the normal form of liveness expressions. An example of using the axiom system is presented in Section 4.4, but first we discuss some aspects of systematically deriving the normal form of a liveness expression. If $x.y$ is a liveness expression, then $\overline{x.y}$ is the complement behaviour. However, one may prefer to divide up the complement into two different equivalence classes based on x and y:

- $\overline{x.*}$: the expression starts with anything other than x.
- $x.\overline{y}$: the expression starts with x, but then continues with anything other than y.

Note that this rewrite is only possible because the following two properties hold:

1. $\epsilon \not\leq x$. As discussed in Section 4.1.1 and proved in Theorem B.2(v), if $\epsilon \leq x$, then $x.* = *$, which implies that all traces are in $x.*$.

2. $\neg(\forall s \in \llbracket x \rrbracket^{\mathcal{T}} \mid (\exists t \in \llbracket \overline{y} \rrbracket^{\mathcal{T}} \mid s^\frown t \in \llbracket x.y \rrbracket^{\mathcal{T}}))$. If this predicate is false, it means that for any trace, s, in x, that there is a trace, t, in $\overline{y}$, such that when concatenated s and t is in $x.y$, which implies that $x.y$ and $x.\overline{y}$ overlap. For example, if $a, b \in \Sigma$, then the liveness expression $\overline{a^+.b}$ would be reduced to $\overline{a^+.*} \mathbin{[\!]} a^+.\overline{b}$. However, the trace $\langle a, a, b\rangle$ is in $a^+.\overline{b}$ (because $\langle a, b\rangle$ is in $\overline{b}$), therefore $\langle a, a, b\rangle$ is in $a^+.b$ and $\overline{a^+.b}$. In Appendix B, we define a relation called *prefixes* such that $x \ \mathit{prefixes}\ \overline{y} \Leftrightarrow \forall s \in \llbracket x \rrbracket^{\mathcal{T}} \mid (\exists t \in \llbracket \overline{y} \rrbracket^{\mathcal{T}} \mid s^\frown t \in \llbracket x.y \rrbracket^{\mathcal{T}})$.

The two sub-expressions, $\overline{x.*}$ and $x.\overline{y}$ are two distinct ways in which an agent playing a role can deviate from its intended behaviour. The complement operators in the sub-expressions can be further expanded using the axiom system until we are left with an expression in normal form. Therefore, we have the following axiom:

$$\neg(x \ \mathit{prefixes}\ \overline{y}) \wedge \epsilon \not\leq x \quad \Rightarrow \quad \overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{y}$$

If either of the premises of this axiom do not hold, a different axiom must be used.

The properties defined in Appendix A comprise the first part of the axiom system. However, these properties are for Kleene algebra, and do not contain axioms for complement expressions. The second part of the axiom system, that which defines axioms for complemented expressions, is defined below.

Axiom System:

(i)	$\overline{\overline{x}}$	$=$	x
(ii)	$x \leq y$	$\Leftrightarrow$	$\overline{y} \leq \overline{x}$
(iii)	$x \mathbin{[\!]} \overline{x}$	$=$	$*$
(iv)	$x \mathbin{[\!]} *$	$=$	$*$
(v)	$*.*$	$=$	$*$
(vi)	$x \sqcap (y \mathbin{[\!]} z)$	$=$	$(x \sqcap y) \mathbin{[\!]} (x \sqcap z)$
(vii)	$x.(y \sqcap z)$	$=$	$x.y \sqcap x.z$
(viii)	$(x \sqcap y).z$	$=$	$x.z \sqcap y.z$
(ix)	$\neg(x \ \mathit{prefixes}\ \overline{y}) \wedge \epsilon \not\leq x$	$\Rightarrow$	$\overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{y}$
(x)	$x \ \mathit{prefixes}\ \overline{y} \wedge \epsilon \not\leq x$	$\Rightarrow$	$\overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{(x.* \mathbin{[\!]} y)}$
(xi)	$\overline{x^*.y}$	$=$	$x^*.(\overline{x.* \mathbin{[\!]} y})$
(xii)	$\overline{x^*}$	$=$	$x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$

In Appendix B, we prove that this axiom system is *sound* — that is, the axioms are valid; and that the axiom system is *complete with respect to normal form* — that is, for any expression, x, its complement, $\overline{x}$, can be reduced to normal form using the axiom system.

Of course, applying the reductions may result in a large expression that is difficult to reason about and maintain, so we introduce some shorthand to help reduce this. For example, reducing the expression $\overline{x^k}$, where k is large, instead of expanding the definition of x^k into k iterations of x and complementing this expression ($\overline{x.x.x...}$), we can use the following theorem (Theorem B.2(vi), proved in Appendix B):

$$k \geq 1 \Rightarrow \overline{x^k} = ([x]^{k\text{-}1}).\overline{x.*} \mathbin{[\!]} x^k.\overline{\epsilon}$$

$[x]^{k-1}$ means between 0 and k-1 iterations of x, which for regular expressions is written $x^{0..k}$. So, the complement of x^k is any number of iterations of x less than k (including 0, which would be x^0, which is equivalent to ϵ), followed by any trace that does not have a trace from x as its prefix, or x^k followed by any non-empty trace.

4.4 Using the Complement Operator

Using the complement operator, for any liveness expression x, we can define the complement liveness expression, that is, the liveness expression that specifies the set of traces that are violations of the expected behaviour. If we reduce the complemented expression into normal form, then we can define disjoint equivalence classes that identify behaviour outside of the behaviour specified by the liveness expression.

As an example, we again use the conference management system from Zambonelli *et al.* [21]. Recall from earlier that the liveness expression for the Reviewer role is as follows.

$$\textsc{Reviewer} = (\mathsf{ReceivePaper.ReviewPaper.SendReviewForm})^{maximum\text{-}number}$$

So, we can define the unexpected behaviour of an agent playing this role, and this role only, with the expression $\overline{\textsc{Reviewer}}$, which can be reduced using the following, which have been proved in the appendices:

$$\neg(x\ prefixes\ \overline{y}) \wedge \epsilon \not\leq x \Rightarrow \overline{x.y} = \overline{x.*} \ [\!]\ x.\overline{y} \tag{4.4}$$

$$k \geq 1 \Rightarrow \overline{x^k} = ([x]^{k\text{-}1}).\overline{x.*} \ [\!]\ x^{k\text{-}1}.\overline{x} \tag{4.5}$$

$$x.(y \ [\!]\ z) = x.y \ [\!]\ x.z \tag{4.6}$$

For readability, we use mn, Recv, Rev, and Send in place of *maximum-number*, ReceivePaper, ReviewPaper, and SendReviewForm respectively.

$$\begin{array}{lll} & \overline{(\mathsf{Recv.Rev.Send})^{mn}} & \\ = & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\overline{\mathsf{Recv.Rev.Send}.*} \ [\!]\ (\mathsf{Recv.Rev.Send})^{mn}.\overline{\epsilon} & \text{Using (4.5)} \end{array}$$

The expression $(\mathsf{Recv.Rev.Send})^{mn}.\overline{\epsilon}$ is not reducible any further, so we continue with the LHS of the choice.

$$\begin{array}{lll} & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\overline{\mathsf{Recv.Rev.Send}.*} & \\ = & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.(\overline{\mathsf{Recv}.*} \ [\!]\ \mathsf{Recv}.\overline{\mathsf{Rev.Send}.*}) & \text{Using (4.4)} \\ = & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.(\overline{\mathsf{Recv}.*} \ [\!]\ \mathsf{Recv}.(\overline{\mathsf{Rev}.*} \ [\!]\ \mathsf{Rev}.\overline{\mathsf{Send}.*})) & \text{Using (4.4)} \\ = & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.(\overline{\mathsf{Recv}.*} \ [\!]\ \mathsf{Recv}.\overline{\mathsf{Rev}.*} \ [\!]\ \mathsf{Recv.Rev}.\overline{\mathsf{Send}.*}) & \text{Using (4.6)} \\ = & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\overline{\mathsf{Recv}.*} \ [\!] & \\ & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\mathsf{Recv}.\overline{\mathsf{Rev}.*} \ [\!] & \\ & [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\mathsf{Recv.Rev}.\overline{\mathsf{Send}.*} & \text{Using (4.6)} \end{array}$$

So far, the reduced expression is as follows:

$$\begin{array}{l} [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\overline{\mathsf{Recv}.*} \ [\!] \\ [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\mathsf{Recv}.\overline{\mathsf{Rev}.*} \ [\!] \\ [\mathsf{Recv.Rev.Send}]^{mn\text{-}1}.\mathsf{Recv.Rev}.\overline{\mathsf{Send}.*} \ [\!] \\ (\mathsf{Recv.Rev.Send})^{mn}.\overline{\epsilon} \end{array}$$

In addition, the expression $[\mathsf{Recv.Rev.Send}]^{mn\text{-}1}$, can be reduced further, using the definition of $[x]$ and x^k, but for readability, we leave it as is.

This now gives us better guidelines as to the different equivalence classes that define the unexpected behaviour. We can read this expression as follows:

"The agent playing this role can violate the behaviour by iterating over the review process i times, where $i < \textit{maximum-number}$, and then by doing one of three things:

1. Participating in any event other than the ReceivePaper protocol;
2. Participating in the ReceivePaper, but then not performing the ReviewPaper action; or
3. Participating in the ReceivePaper and then performing the ReviewPaper action, but not participating in the SendReviewForm protocol.

Alternatively, the agent can violate the behaviour by iterating over the review process *maximum-number* times, and then performing anything other than ϵ (in other words, performing anything)."

Dividing the complement of a liveness expression into equivalence classes gives developers a systematic way of dealing with exceptions. In programming logic, the programmer can treat each equivalence class as a separate exception, and may have different methods for dealing with them. For example, in the conference management system, if an agent does not participate in the ReceivePaper protocol, then we can find another agent to review the paper. However, if they accept a proposal to review a paper, but then do not follow through, we may have to take further action, such as obtaining another copy of the paper.

5 Formalising Roles and Liveness Rules

In addition to specifying in which events are role can participate, Gaia permits one to specify the rules about a role, and the relationships between roles. For example, if we have two roles R and Q, we can specify a liveness rule that states that role R must be played 3 times before role Q can be played: $R^3 \rightarrow Q$

Not only does this imply that we need to formalise new operators, such as '$\rightarrow$', but we also need to provide a semantics for referencing roles via names. That is, the name references R and Q above represent sets of traces that have been declared previously.

In this section, we define a denotational semantics for roles and liveness rules. Throughout this section, L and M represent liveness rules, and R and Q represent role names.

5.1 Role Definitions

To reference roles that are declared in a previous part of a Gaia specification, we are required to maintain a mapping between role names and the liveness expression that they represent, or more specifically, the traces specified by the liveness expression. To do this, we introduce a set called $Name$, which specifies the set of all role names, and introduce $Roles$, the set of partial functions from names to sets of traces:

$$Roles == Name \nrightarrow \mathcal{P}(\Sigma^*)$$

The semantics of a role definition, $R = E$, in which R is the role name and E the liveness expressions for that role, is written $[[R = E]]^{\mathcal{R}}$, such that $[[R = E]]^{\mathcal{R}} \in Roles$. It defines a set of roles and the traces associated with each role. Formally, we define this as the function mapping the role name to its traces:

$$[\![R = E]\!]^{\mathcal{R}} \quad \widehat{=} \quad \{R \mapsto [\![E]\!]^{\mathcal{T}}\} \quad \textbf{where } R \in \mathit{Name} \land E \in \Omega$$

The expression $[\![\mathrm{R} = a.b.c]\!]^{\mathcal{R}}$ reduces to $\{\mathrm{R} \mapsto \{\langle a, b, c\rangle\}\}$. Applying R to this function will result in $\{\langle a, b, c\rangle\}$, which is the behaviour of R.

However, we must also distinguish the traces that a role R can play from the set of traces that Q can play. For example, in the rule $R \parallel Q$, which specifies that roles R and Q act concurrently, if R and Q share some of the same events in their alphabet, then we will not be able to say whether the occurrence of a shared event is from R or Q. Therefore, we prefix the names of events with their role name. This requires a change to the definition of $[\![R = E]\!]^{\mathcal{R}}$ above:

$$[\![R = E]\!]^{\mathcal{R}} \quad \widehat{=} \quad \{R \mapsto \mathit{label}([\![E]\!]^{\mathcal{T}}, R)\} \quad \textbf{where } R \in \mathit{Name} \land E \in \Omega$$

in which $\mathit{label} \in (\mathcal{P}(\Sigma^*) \times \mathit{Name}) \rightarrow \mathcal{P}(\Sigma^*)$ is a function that adds a prefix to every event, such that the event e in a role R would be $R.e$. The events $R.e$ and $Q.e$ are performed by different roles, therefore, in the rule $R \parallel Q$, an occurrence of e promoted by role R will be distinguished by one from Q because of its prefix.

A Gaia specification would typically contain several role definitions, which are related. The semantics of a series of n role definitions in a document, $\{R_1 = E_1, \ldots, R_n = E_n\}$, is the union of the functions all of the role definitions:

$$\begin{aligned}[\![R_1 = E_1; \ldots; R_n = E_n]\!]^{\mathcal{R}} \quad \widehat{=} \quad & [\![R_1 = E_1]\!]^{\mathcal{R}} \cup \ldots \cup [\![R_n = E_n]\!]^{\mathcal{R}} \\ & \textbf{where } R_1, \ldots, R_n \in \mathit{Name} \land E_1, \ldots, E_n \in \Omega\end{aligned}$$

For example, take the following two role definitions:

$\mathrm{R} = a.b.c$
$\mathrm{Q} = d.e.f$

The definition, $[\![\mathrm{R} = a.b.c; \mathrm{Q} = d.e.f]\!]^{\mathcal{R}}$ becomes $[\![\mathrm{R} = a.b.c]\!]^{\mathcal{R}} \cup [\![\mathrm{Q} = d.e.f]\!]^{\mathcal{R}}$ which is reduced further to $\{R \mapsto \{\langle R.a, R.b, R.c\rangle\}\} \cup \{Q \mapsto \{\langle Q.d, Q.e, Q.f\rangle\}\}$, and then finally to $\{R \mapsto \{\langle R.a, R.b, R.c\rangle\}, Q \mapsto \{\langle Q.d, Q.e, Q.f\rangle\}\}$. This now gives us a function mapping the roles in the specification to their possible behaviour.

5.2 Parameterised Role Definitions

Gaia roles can be parameterised such that instantiating the parameters defines different roles. For example, Zambonelli *et al.* [21] define a role *Stage*[i], in a pipeline manufacturing process. At the i_{th} stage of the processing pipeline, the *Stage* role must monitor the flux of incoming items from the *Stage* role at the i-1_{th} stage, reducing the speed of its production if the items are arriving slower than expected. This is defined as follows:

$$\textsc{Stage}[i] = (\mathsf{MonitorFlux[i\text{-}1]}^*.\mathsf{ReduceSpeed}.\mathsf{OKReduceSpeed})^*$$

Therefore, Stage[i] actually specifies a *set* of roles, rather than just a single role. The set of roles consists of the roles named Stage[1], Stage[2], Stage[3] …, etc., in which the event MonitorFlux[i-1] is instantiated as MonitorFlux[0], MonitorFlux[1], MonitorFlux[2], …, etc., respectively.

In our formalism, we allow parameters to be variables ranged over natural numbers. Instantiated role names, for example, Stage[1], are treated as role name; that is, Stage[i] $\in$ *Name*; and instantiated events, such as MonitorFlux[1] are treated as atomic events; that is MonitorFlux[1] $\in \Sigma$.

Therefore, the definition of a role, $R[N_1, \ldots, N_n] = E$, is defined as a set of roles, in which all parameters are instantiated with all possible natural numbers, and all occurrences of the parameters substituted in the expression E:

$$[\![R[N_1, \ldots, N_n] = E]\!]^{\mathcal{R}} \quad \widehat{=} \quad \bigcup_{i_1, \ldots, i_n \geq 1} [\![R[i_1, \ldots, i_n] = E[N_1 \mapsto i_1, \ldots, N_n \mapsto i_n]]\!]^{\mathcal{R}}$$

in which $E[N \mapsto i]$ represents the expression E with all occurrences of N substituted with i. So, this definition expands all permutations of parameters, for example, the definition of $\textsc{Stage}[i]$ above would be:

$[\![\textsc{Stage}[1] = (\mathsf{MonitorFlux[0]}^*.\mathsf{ReduceSpeed.OKReduceSpeed})^*]\!]^{\mathcal{R}} \cup$
$[\![\textsc{Stage}[2] = (\mathsf{MonitorFlux[1]}^*.\mathsf{ReduceSpeed.OKReduceSpeed})^*]\!]^{\mathcal{R}} \cup$
$[\![\textsc{Stage}[3] = (\mathsf{MonitorFlux[2]}^*.\mathsf{ReduceSpeed.OKReduceSpeed})^*]\!]^{\mathcal{R}} \cup$
...

So, $[\![R[N_1, \ldots, N_n] = E]\!]^{\mathcal{R}}$ specifies an infinite set of role definitions, which can now be referenced using the instantiated name, for example, $\textsc{Stage}[1]$.

5.3 Liveness Rules

Gaia liveness rules are similar to liveness expressions, except that they specify the behaviour of many roles in a system. For example, if we have the roles R and Q, then we can specify that role R must be played 3 times before role Q can be played: $R^3 \rightarrow Q$. This constrains the behaviour of a role when placed within this system.

5.3.1 Liveness Rule Operators

The semantics of the liveness rules is given using the function $[\![L]\!]^{\mathcal{L}} \in Roles \rightarrow \mathcal{P}(\Sigma^*)$, in which L is a liveness rule. Therefore, in the context of a set of role definitions, a rule produces a set of traces that specify the possible behaviour of the system.

The definition of the '$\rightarrow$' operator is similar to the liveness expression operator '.', however, we have to take into account that the arguments may contain *references* to liveness expressions, not just explicit liveness expressions. Subsequently, the semantics of a liveness rule is given in the context of a set of role definitions (from the set *Roles*). So, the definition of the '$\rightarrow$' operator is as follows:

$$[\![L \rightarrow M]\!]^{\mathcal{L}} \quad \widehat{=} \quad \lambda r : Roles \bullet \{ s ^\frown t \mid s \in [\![L]\!]^{\mathcal{L}}(r) \wedge t \in [\![M]\!]^{\mathcal{L}}(r) \}$$

This states that the set of traces produced by $L \rightarrow M$ in the context of a role r, is the set of traces resulting from the concatenation of every trace in $[\![L]\!]^{\mathcal{L}}(r)$ (the result of applying r to the function given by $[\![L]\!]^{\mathcal{L}}$) with every trace in $[\![M]\!]^{\mathcal{L}}(r)$ (the result of applying r to the function given by $[\![M]\!]^{\mathcal{L}}$).

One can see that the definition is similar to the definition of the '.' operator in Section 3. In fact, many of the definitions of operators for liveness rules have a counterpart definition for liveness expressions, for example, $L \parallel M$ for interleaving. Table 3 shows the operators for liveness rules, and their equivalent in liveness expressions. From this table and the definitions in Section 3, readers should be able to infer the formal definitions of the operators.

This is useful because, in the context of a set of role definitions, a liveness rule can be treated as a liveness expression, which implies that the theorems for liveness expressions can be used on

Operator	Counterpart
$L \rightarrow M$	$x.y$
$L \vee M$	$x \; [] \; y$
$L \wedge M$	$x \sqcap y$
L^*	x^*
L^+	x^+
L^k	x^k
$[L]$	$[x]$
$L \parallel M$	$x \parallel y$
$\neg L$	$\overline{x}$
$L^{k..m}$	$x^k.[x]^m$

Table 3: Operators for Liveness Rules

liveness rules also, including Kleene algebra axioms.

5.3.2 Role References

The only definition that does not have a counterpart in liveness expressions, is a reference to a role name, because liveness expressions do not support references. The semantics for a reference explicitly makes use of the context in which it is used. The set of traces of a reference to a role, R, in the context of a roles function, r, is the value of that role in r. Formally:

$$[\![R]\!]^{\mathcal{L}} \quad \widehat{=} \quad \lambda r : \mathit{Roles} \bullet r(R) \quad \textbf{where } R \in \mathit{Name}$$

So, if we have a role definition R = $a.b.c$, then the behaviour of R is derived by applying the function that makes up R = $a.b$ to a reference to R:

$$[\![\mathrm{R} = a.b.c]\!]^{\mathcal{L}} \ (\mathrm{R}) = \{\langle R.a, R.b, R, c\rangle\}$$

5.4 Synchronous Behaviour

In this section, we propose an alternative semantics for the interleaving operator. This allows events performed by different roles to be synchronous. That is, if R plays event a, and Q plays event b, they can occur at the same time. This is especially useful for modelling interaction between two roles. For example, if role Q is to wait for role R to send a specific message before it starts, then the events $R.SendMessage$ and $Q.ReceiveMessage$ are the same event: they both model the message being sent from R to Q. Therefore, they can occur at the same time.

The semantics of the liveness rule $L \parallel M$ is every trace in L synchronously interleaved with every trace in M. By this, we mean that we model every case in which two events can occur either one after the other, or at the same time. For example, take the following role definitions:

$$R = a$$
$$Q = b$$

The synchronous interleaving of R and Q is $\{\langle R.a, Q.b\rangle, \langle Q.b, R.a\rangle, \langle R.a \leftrightarrow Q.b\rangle\}$, in which $R.a \leftrightarrow Q.b$ represents the event of $R.a$ and $Q.b$ occurring simultaneously, and is equivalent to the event $Q.b \leftrightarrow R.a$.

Formally, we define the the operator semantics, in the context of the set of role definitions, r, as being every trace from L synchronised with every trace from M:

$$[\![L \parallel M]\!]^{\mathcal{L}} \quad \widehat{=} \quad \lambda r : \mathit{Roles} \bullet \{s \mid (\exists t \in [\![L]\!]^{\mathcal{L}}(r), u \in [\![M]\!]^{\mathcal{L}}(r) \mid s\ \mathit{synchronises}(u, t))\}$$

The relation *synchronises* is defined in a similar manner to *interleaves*, except the is a fourth condition under which two sequences are synchronously interleaved: when the events at the head of the sequences occur synchronously, and the tail of the traces are synchronously interleaved:

L1 $\langle\,\rangle\ \mathit{synchronises}(t, u) \Leftrightarrow t = u = \langle\,\rangle$

L2 $s\ \mathit{synchronises}(t, u) \Leftrightarrow s\ \mathit{synchronises}(u, t)$

L3 $\langle a\rangle^\frown s\ \mathit{synchronises}(t, u) \Leftrightarrow$
$\quad t \neq \langle\,\rangle \wedge t_0 = a \wedge s\ \mathit{synchronises}(t', u) \vee$
$\quad u \neq \langle\,\rangle \wedge u_0 = a \wedge s\ \mathit{synchronises}(t, u')$

L4 $\langle a{\leftrightarrow}b\rangle^\frown s\ \mathit{synchronises}(t, u) \Leftrightarrow$
$\quad t \neq \langle\,\rangle \wedge u \neq \langle\,\rangle \wedge$
$\quad (t_0 = a \wedge u_0 = b \vee t_0 = b \wedge u_0 = a) \wedge$
$\quad s\ \mathit{synchronises}(t', u')$

Returning to our example of R sending Q a message, then the new definition of interleaving is not sufficient to enforce that the events occur simultaneously, because interleaved behaviour is also permitted. To specify that any occurrence of $R.\mathit{SendMessage}$ is synchronised with $Q.\mathit{ReceiveMessage}$, we use the $\bullet$ operator, followed by a set of synchronised events. That is:

$$R \parallel Q \bullet \{R.\mathit{SendMessage} {\leftrightarrow} Q.\mathit{ReceiveMessage}\}$$

Formally, we define this operator as follows:

$$[\![L \bullet \{a_1, \ldots, a_n\}]\!]^{\mathcal{L}} \quad \widehat{=} \quad \lambda r : \mathit{Roles} \bullet \{s \in [\![L]\!]^{\mathcal{L}}(r) \mid s\ \mathit{contains\_only}\{a_1, \ldots, a_n\}\}$$

In which $\mathit{contains\_only} \in \mathcal{P}(\Sigma^* \times \mathcal{P}(\Sigma))$ is a relation defined as follows:

$$\mathit{contains\_only}(s, pa) \Leftrightarrow$$
$$(\forall a_1 \in \mathit{ran}(s) \mid (\forall a_2 \in pa \mid \mathit{events}(a_1) \cap \mathit{events}(a_2) = \emptyset \Rightarrow \mathit{events}(a_2) \subseteq \mathit{events}(a_1)))$$

In which $\mathit{events}(a)$ returns the non-synchronised events in a, for example, $\mathit{events}(a) = \{a\}$ and $\mathit{events}(a{\leftrightarrow}b{\leftrightarrow}c) = \{a, b, c\}$. So, *contains_only* specifies that for any trace, s, then for every event in s, if any of its sub-events are referenced in the set of synchronised events, then they must be synchronised with the events related by $\leftrightarrow$ in that reference.

This operator is expressive enough to specify that a must always occurs by itself: $L \bullet \{a\}$. We also note that the expression $L \bullet \{a{\leftrightarrow}b{\leftrightarrow}c\}$ is equivalent to $L \bullet \{a{\leftrightarrow}b, b{\leftrightarrow}c\}$.

The downside of this alternative definition is that the theorems about the liveness expression interleaving are not applicable, and the more complex definition would make any theorems more difficult to derive and prove.

5.5 Combining Roles and Rules

So far, we have presented the semantics of role definitions, and of rules with the context of a set of role definitions. Defining the semantics of a system comprising a set of role definitions and a set of rules is straightforward. If we have a set of role definitions, $R_1 = E_1, \ldots, R_n = E_n$, and the liveness rules quantified over those role definitions, L, then the set of traces of events that satisfy that system defined by applying the function defined by the roles to the rules. Formally:

$$[\![R_1 = E_N; \ldots; R_n = E_n; L]\!]^{\mathcal{T}} \quad \widehat{=} \quad [\![L]\!]^{\mathcal{L}} \; ([\![R_1 = E_1, \ldots, R_n = E_n]\!]^{\mathcal{R}})$$

To help with the understanding of this, we return to an example. First, we take the definition of rules R and Q from Section 5.1.

R $= a.b.c$
Q $= d.e.f$

Recall that the function produced by these definitions is the following:

$$\{R \mapsto \{\langle R.a, R.b, R.c \rangle\}, Q \mapsto \{\langle Q.d, Q.e, Q.f \rangle\}\}$$

which we will abbreviate to r. The liveness expression rule $R^3 \to Q$ specifies that the role R must be play 3 times before role Q can be played. So, in the context of the roles R and Q, this is equivalent to the following:

$$\begin{array}{lll} & [\![R^3 \to Q]\!]^{\mathcal{L}}(r) & \\ \equiv & \{s ^\frown t \mid s \in [\![R^3]\!]^{\mathcal{L}}(r) \wedge t \in [\![Q]\!]^{\mathcal{L}}(r)\} & \text{Using definition of } \to \\ \equiv & \{s ^\frown t \mid s \in [\![R \to R \to R]\!]^{\mathcal{L}}(r) \wedge t \in [\![Q]\!]^{\mathcal{L}}(r)\} & \text{Using definition of } L^k \end{array}$$

Applications of '$\to$' are recursively unfolded, until we need to unfold the references to R and Q. To unfold the reference to R, we use the definition from above, which in the context of r, is $r(R)$, which is the set of traces $\{\langle R.a, R.b, R.c \rangle\}$. With $R \to R \to R$ being reduced, we have that R^3 produces the traces $R.a$ to $R.c$ 3 times, and from the definition of Q, we have its reference unfolding to $\{\langle Q.d, Q.e, Q.f \rangle\}$. So, the above expression is reduced to the following:

$$\{s ^\frown t \mid s \in \{\langle R.a, R.b, R.c, R.a, R.b, R.c, R.a, R.b, R.c \rangle\} \wedge t \in \{\langle Q.d, Q.e, Q.f \rangle\}\}$$

and the resulting set of traces of the liveness rule under the context r is:

$$\{\langle R.a, R.b, R.c, R.a, R.b, R.c, R.a, R.b, R.c, Q.d, Q.e, Q.f \rangle\}$$

This states that R must do a, b, c 3 times, then Q must do d, e, f once, which is the specification that we wanted.

6 Conclusions

In this paper, we have presented a formal semantics for Gaia liveness expressions and liveness rules, and discussed a sound and complete axiom system for these. While Gaia is designed to be an informal development methodology, our experience indicates that there are cases in which a formal semantics for Gaia is useful. Our formalism for liveness expressions and rules in Gaia expand the scope within which Gaia can be used for development, while at the same time, maintaining the benefits of Gaia in earlier stages of development by preserving the existing syntax.

The semantics is given such that a liveness expression defines the set of possible traces of events that an agent playing a role participates in or performs, and that a liveness rule defines the set of possible traces of events that a set of *interacting* roles participate in or perform. Our formalism of liveness expressions has been proven to satisfy the properties of a Kleene algebra, allowing us to use the results of other work on Kleene algebras.

In addition to formalising the liveness rules and expressions as given by Wooldridge *et al.* [20], we introduce a new operator for defining the complement of these expressions. This operator gives developers more flexibility for writing and reasoning about roles and their interactions in Gaia. For example, using the complement operator, one can provide an easy reference to the behaviour that we do not want a role to exhibit.

An axiom system for complemented expression has been presented, and proved to be sound and complete with respect to our notion of normal form. These axioms allow us to do such things as systematically derive disjoint equivalence classes from an expression, as is demonstrated in the paper, to help us reason about the behaviour of roles. A specific use of this is to help identify the different equivalence classes of exceptional behaviour that an agent may exhibit when playing a role.

The formalism presented in this paper has been applied to the PIPS[4] system. Gaia was used for analysis and specification of a subsystem of PIPS, and, using the formalism presented in this paper, the role specifications, with their formally specified liveness expressions, were used to guide the design and development of some agents in the system, as well as to identify exceptional behaviour.

While we believe this formalism plugs a hole that has existed in Gaia since its origin, we do not believe that this paper completes the formalisation of roles in Gaia. In Gaia, the environment is represented as a collection of *abstract computational resources*, which are made available to the agents, and are modelled using mathematical notation such as variables and sets. A state-based formal language, such as Z [18], could be used to model this completely and unambiguously. However, a further step in Gaia would be a formalism for mapping the effect that atomic events have on the state of the environment; for example, the atomic action ReviewPaper would change the status of the paper in the environment to "reviewed". At present, mappings between the environment and the effect events have on the environment are informal, and the effects of events on the environment can be ambiguous and difficult to determine.

Additional further work is required to exploit the power of the formalism in this paper. Now that a formalism of Gaia liveness expression exists, and a new operator has been added to the language, it is necessary to look at how this can help us. One aspect we are looking at is the types of properties that could be verified using this formalism. While most properties are specific to the application domain, guidelines can be provided for certain activities, such as verifying that certain events occur before others, or that certain events only occur in the absence of others. Other work we are pursuing includes guidelines for identifying and documenting the exceptional behaviour of roles using the formalism and axioms, as well as to create disjoint partitions for test-case generation, similar to the techniques used in specification-based testing.

Acknowledgments:

The authors are grateful for financial support from the European Commission's Information Society Technologies Programme through the Project *Personalized Information Platform for Life and Health Services* (IST-FP6-507019) and the UK Engineering and Physical Sciences Research Council (EPSRC), through the Project *Market-Based Control of Complex Computational Systems* (GR/T19742/01).

[4]`http://www.pips.eu.org/`.

References

[1] K. Aboul-Hosn and D. Kozen. KAT-ML: An interactive theorem prover for Kleene Algebra with Tests. In B. Konev and R. Schmidt, editors, *Proceedings of 4th International Workshop on the Implementation of Logics*, pages 2–12, 2003.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[3] D. Coleman, P. Arnold, S. Bodoff, D. Dollin, H. Gilchrist, F. Hayes, and P. Jeremas. *Object-Oriented Development: The FUSION Method*. Prentice-Hall International, Hemel Hampstead, U.K., 1994.

[4] J. Friedl. *Mastering Regular Expressions*. O'Reilly: Sebastopol, USA, 2nd edition, 2002.

[5] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[6] T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 3–10. ACM Press, 2002.

[7] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 56–71. Springer-Verlag: Berlin, Germany, 1996.

[8] D. Kozen. On Kleene algebras and closed semirings. In B. Rovan, editor, *Proceedings of Mathematical Foundations of Computer Science*, volume 452 of *LNCS*, pages 26–47. Springer, 1990.

[9] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, May 1994.

[10] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, (3):427–443, 1997.

[11] T. Miller and P. McBurney. A formal semantics for Gaia liveness rules and expressions. Technical Report UCLS-05-012, Department of Computer Science, University of Liverpool, 2005.

[12] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag: Berlin, Germany, 1980.

[13] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press: Cambridge, UK, 1999.

[14] L. Padgham and M. Winikoff. Prometheus: A pragmatic methodology for engineering intelligent systems. In *Proceedings of the Workshop on Agent-Oriented Methodologies at OOPSLA*, 2002.

[15] A. Perini, M. Pistore, M. Roveri, and A. Susi. Agent-oriented modeling by interleaving formal and informal specification. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering, 4th International Workshop*, pages 36–52. Springer, 2003.

[16] PIPS Project. Specification and first prototype of agent profile and virtual environment. Deliverable 4.3.1, June 2005. `http://www.pips.eu.org/`.

[17] A. Rao and M. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian Artificial Intelligence Institute, Melbourne, Australia, June 1995.

[18] J. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall: Hertfordshire, UK, 2nd edition, 1992.

[19] M. Wood and S. DeLoach. An overview of the multiagent systems engineering methodology. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, volume 1957 of LNCS, pages 127–141. Springer: Berline, Germany, 2000.

[20] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[21] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.

A Liveness Expressions form a Kleene Algebra

In this appendix, we prove that the formalised liveness expressions defined in Section 3 form a Kleene algebra.

A *Kleene algebra* is defined as a structure $\mathcal{R}$ with binary operations $+$ and $.$, unary operator *, and constants 0 and 1 such that $(\mathcal{R}, +, ., 0, 1)$ form a *idempotent semiring*. That is, the following properties hold:

1. $(\mathcal{R}, +)$ is a commutative monoid with identity 0. That is, for all x, y, and z in $\mathcal{R}$:
    - $(x + y) + z = x + (y + z)$ (+ is associative)
    - $x + 0 = x = 0 + x$ (0 is an identity for +)
    - $x + y = y + x$ (+ is commutative)
    - $x + x = x$ (+ is idempotent)
2. $(\mathcal{R}, .)$ is a monoid with identity 1. That is, for all x, y, and z in $\mathcal{R}$:
    - $(x.y).z = x.(y.z)$ (. is associative)
    - $1.x = x = x.1$ (1 is an identity for .)
3. . distributes over +. That is, for all x, y, and z in $\mathcal{R}$:
    - $x.(y + z) = x.y + x.z$ (. left-distributes over +)
    - $(x + y).z = x.z + y.z$ (. right-distributes over +)
4. 0 annihilates . on $\mathcal{R}$. That is, for all x in $\mathcal{R}$:
    - $x.0 = x = 0.x$

Additionally, the * operator satisfies the following properties for all x and y in $\mathcal{R}$:

- $1 + x.x^* \leq x^*$
- $1 + x^*.x \leq x^*$
- $x.y \leq y \Rightarrow x^*.y \leq y$
- $y.x \leq y \Rightarrow y.x^* \leq y$

in which the relation $\leq$ defines a partial order on $\mathcal{R}$, such that for all x, y in $\mathcal{R}$:

$x \leq y \Leftrightarrow x + y = y$

In this section, we prove that $(\Omega, [], ., \emptyset, \epsilon)$, as defined in Section 3, and in which Ω represents the set of all liveness expressions, form a Kleene algebra.

We use the following rules throughout these appendices:

$$a \notin \{d \mid p\} \Leftrightarrow a \in \{d \mid \neg p\} \tag{A.1}$$

$$\{d \mid d \in X\} = X \tag{A.2}$$

$$\{d \mid p_1\} \cap \{d \mid p_1\} = \{d \mid p_1 \wedge p_2\} \tag{A.3}$$

$$\{d \mid p_1\} \cup \{d \mid p_1\} = \{d \mid p_1 \vee p_2\} \tag{A.4}$$

(A.1) and (A.2) hold due to the Law of Excluded Middle, while (A.3) and (A.4) are straightforward from the definition of $\cap$ and $\cup$ respectively. We use the labels on the right to reference these rules.

Theorem A.1. $(\Omega, [], ., \emptyset, \epsilon)$ *forms a Kleene algebra*

Proof. The prove that our formalism of liveness expressions forms a Kleene algebra, we propose and prove lemmas regarding the operators and their semantics, which related to the definition of a Kleene algebra.

Lemma A.2. $(\Omega, [])$ *is a Commutative Monoid with Identity* $\emptyset$

Proof. This trivial proof is omitted. A sketch of the proof involves expanding the definitions of $[]$ and $\emptyset$, and observing that $(\Sigma^*, \cup)$ forms a commutative monoid with identity $\emptyset$. $\square$

Lemma A.3. $(\Omega, .)$ *is a Monoid with Identity* ϵ

Proof. To prove this, we need to prove the following two properties:

(i) $(x.y).z = x.(y.z)$
(ii) $\epsilon.x = x = x.\epsilon$

To prove (i), we expand the definition of ., leaving us to show the following:

$$\{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in \{u^\frown v \mid u \in [\![y]\!]^{\mathcal{T}} \wedge v \in [\![z]\!]^{\mathcal{T}}\}\} = \\ \{s^\frown t \mid s \in \{u^\frown v \mid u \in [\![x]\!]^{\mathcal{T}} \wedge v \in [\![y]\!]^{\mathcal{T}}\} \wedge t \in [\![z]\!]^{\mathcal{T}}\}$$

We reduce the LHS of this equality using (A.2) and substituting $u^\frown v$ for t

$$\begin{aligned} & \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in \{u^\frown v \mid u \in [\![y]\!]^{\mathcal{T}} \wedge v \in [\![z]\!]^{\mathcal{T}}\}\} \\ \equiv \quad & \{s^\frown(u^\frown v) \mid s \in [\![x]\!]^{\mathcal{T}} \wedge u \in [\![y]\!]^{\mathcal{T}} \wedge v \in [\![z]\!]^{\mathcal{T}}\} \end{aligned}$$

We do the same for the RHS of the equality, except substituting $u^\frown v$ for s. We are left with the following:

$$\{(u^\frown v)^\frown t \mid u \in [\![x]\!]^{\mathcal{T}} \wedge v \in [\![y]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\} = \\ \{s^\frown(u^\frown v) \mid s \in [\![x]\!]^{\mathcal{T}} \wedge u \in [\![y]\!]^{\mathcal{T}} \wedge v \in [\![z]\!]^{\mathcal{T}}\}$$

From the associativity of $^\frown$, this predicate holds trivially.

To prove (ii), we expand the definitions of ϵ and ., leaving us to show the following:

$$\{s^\frown t \mid s \in \{\langle\rangle\} \wedge t \in [\![x]\!]^{\mathcal{T}}\} = [\![x]\!]^{\mathcal{T}} = \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in \{\langle\rangle\}\}$$

The first equality, we see that in each case that $s^\frown t$, s is empty, therefore, $s^\frown t = t$ and the set resolves to $\{t \mid t \in [\![x]\!]^{\mathcal{T}}\}$, which is simply $[\![x]\!]^{\mathcal{T}}$. Similarly, for the second equality except t is empty. Therefore, the lemma is valid. $\square$

Lemma A.4. . *distributes over* $[]$

Proof. To prove this, we have to prove the following:

(i) $x.(y \mathbin{[]} z) = x.y \mathbin{[]} x.z$
(ii) $(x \mathbin{[]} y).z = x.z \mathbin{[]} y.z$

To prove (i), we expand the definitions of . and $[]$, leaving us to show the following

$$\{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}} \cup [\![z]\!]^{\mathcal{T}}\} = \\ \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}\} \cup \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\}$$

We reduce the RHS of the equality.

$$\begin{array}{lll} & \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}\} \cup \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\} & \\ \equiv & \{s^\frown t \mid (s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}) \vee (s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}})\} & \text{using (A.4)} \\ \equiv & \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge (t \in [\![y]\!]^{\mathcal{T}} \vee t \in [\![z]\!]^{\mathcal{T}})\} & \text{using distributivity of } \vee \text{ over } \wedge \\ \equiv & \{s^\frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}} \cup [\![z]\!]^{\mathcal{T}}\} & \text{using (A.1) and (A.4)} \end{array}$$

This is trivially equivalent to the LHS. The proof for (ii) is similar to the proof for (i), so is omitted. □

Lemma A.5. *$\emptyset$ annihilates .*

Proof. To prove this, we have to show $\emptyset.x = \emptyset = x.\emptyset$. First, we show $\emptyset.x = \emptyset$. Expanding the definitions of . and $\emptyset$ leaves us to prove the following:

$$\{s^\frown t \mid s \in \emptyset \wedge t \in [\![x]\!]^{\mathcal{T}}\} = \emptyset$$

The predicate $s \in \emptyset$ is false for all s, therefore, the LHS defines the empty set.

Similarly, $[\![x.\emptyset]\!]^{\mathcal{T}}$ is empty, therefore $\emptyset.x = \emptyset = x.\emptyset$. □

Lemma A.6. *The following properties hold:*

- $\epsilon + x.x^* \leq x^*$
- $\epsilon + x^*.x \leq x^*$
- $x.y \leq y \Rightarrow x^*.y \leq y$
- $y.x \leq y \Rightarrow y.x^* \leq y$

Proof. Kozen [8] states that the *-*continuity condition* implies these four properties. The *-continuity condition is specified as follows:

$$x.y^*.z = \bigcup_{n \geq 0} x.y^n.z$$

So, instead of proving each of these four properties, we instead prove that the *-*continuity condition* holds for our semantics.

Expanding the definition of $\bigcup$, we expand the RHS to the following:

$$[\![x.y^0.z]\!]^{\mathcal{T}} \cup [\![x.y^1.z]\!]^{\mathcal{T}} \cup [\![x.y^2.z]\!]^{\mathcal{T}} \cup [\![x.y^3.z]\!]^{\mathcal{T}} \cup \ldots$$

In which the ellipsis $(\ldots)$ indicates that the expressions continue infinitely, incrementing k in y^k each time. From the definition of $[\!]$, this is equivalent to the following:

$$x.y^0.z \;[\!]\; x.y^1.z \;[\!]\; x.y^2.z \;[\!]\; x.y^3.z \;[\!]\; \ldots$$

We use Lemma A.4 on this expression:

$$x.(y^0 \;[\!]\; y^1 \;[\!]\; y^2 \;[\!]\; y^3 \;[\!]\; \ldots).z$$

From the definition of *, this expression is equivalent to:

$x.y^*.z$

which is equivalent to the LHS. Therefore, the *-continuity condition holds, implying the four properties over * hold, and our formalism of liveness expressions forms a Kleene algebra. $\square$

Theorem A.1 follows directly from Lemmas A.2—A.6, therefore, our formalism of liveness expressions forms a Kleene algebra. $\square$

A.1 Elementary Properties of Kleene Algebras

In this section, list some elementary properties of Kleene algebras. These hold for all Kleene algebras, and from our proof in this section that our definition of liveness expressions form a Kleene algebra, they can be used to rewrite our Gaia liveness expressions.

In any Kleene algebra, $(\mathcal{R}, +, ., 0, 1)$, with partial order relation $\leq$, we have the following properties for all x, y, and z in $\mathcal{R}$.

$$\begin{array}{rcl}
x \leq y & \Rightarrow & x.z \leq y.z \\
x \leq y & \Rightarrow & z.x \leq z.y \\
x \leq y & \Rightarrow & x + z \leq y + z \\
x \leq y & \Rightarrow & x^* \leq y^* \\
1 & \leq & x^* \\
x & \leq & x^* \\
1 + x + x^*.x^* & = & x^* \\
(x^*)^* & = & x^* \\
x^*.x^* & = & x^* \\
x^*.x & = & x.x^* \\
0^* & = & 1 \\
1 + x.x^* & = & x^* \\
1 + x^*.x & = & x^* \\
y + x.z \leq z & \Rightarrow & x^*.y \leq z \\
y + z.x \leq z & \Rightarrow & y.x^* \leq z \\
x.z = z.y & \Rightarrow & x^*.z = z.y^* \\
(x.y)^*.x & = & x.(y.x)^* \\
(x + y)^* & = & x^*.(y.x^*)^*
\end{array}$$

B Soundness and Completeness of Axiom System

In this appendix, we prove the soundness and completeness of the axiom system introduced in Section 4.3. These axioms are applicable in general to Kleene algebra that have a complement operator. We prove the soundness of these axioms, and their completeness with respect to normal form; that is, we prove that all of the axioms are valid, and that any expression can be reduced to normal form using these axioms and the properties of Kleene algebra defined in Appendix A.

There are several assumptions made about the expressions in the axioms of the section:

- No redundant operators are used. That is, any reference to a redundant operator us rewritten in terms of primitive operators before the rules are applied.
- Expressions not containing complements are written in *normal form.*
- Expressions are reduced as far as possible using the rules in Appendix A. For example, $\epsilon.x$ becomes x, and $x^*.x$ becomes $x.x^*$. As a result, expressions of the form x^*, x^+, and x^k, x is does not contain the empty trace ($\epsilon \not\leq x$), and are not applications of the iteration operator; that is, it is not of the form $(x^*)^*$, $(x.x^+)^*$, etc.

We define a binary relation called *prefixes*:

$$x \; prefixes \; \overline{y} \Leftrightarrow \forall s \in [\![x]\!]^{\mathcal{T}} \mid (\exists t \in [\![\overline{y}]\!]^{\mathcal{T}} \mid s \frown t \in x.y)$$

It is the case that $x \; prefixes \; \overline{y}$ only holds if x is an infinitely iterated expression, such as z^* or $z.z^*$, and for every trace in x, at least one trace in $\overline{y}$ has its prefix from x, and the rest of the trace from y.

B.1 Soundness

Recall from Section 4.3, the following axiom system for complemented liveness expressions:

$$\begin{array}{lrcl}
\text{(i)} & \overline{\overline{x}} & = & x \\
\text{(ii)} & x \leq y & \Leftrightarrow & \overline{y} \leq \overline{x} \\
\text{(iii)} & x \mathbin{[\!]} \overline{x} & = & * \\
\text{(iv)} & x \mathbin{[\!]} * & = & * \\
\text{(v)} & *.* & = & * \\
\text{(vi)} & x \sqcap (y \mathbin{[\!]} z) & = & (x \sqcap y) \mathbin{[\!]} (x \sqcap z) \\
\text{(vii)} & x.(y \sqcap z) & = & x.y \sqcap x.z \\
\text{(viii)} & (x \sqcap y).z & = & x.z \sqcap y.z \\
\text{(ix)} & \neg(x \; prefixes \; \overline{y}) \wedge \epsilon \not\leq x & \Rightarrow & \overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{y} \\
\text{(x)} & x \; prefixes \; \overline{y} \wedge \epsilon \not\leq x & \Rightarrow & \overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{(x.* \mathbin{[\!]} y)} \\
\text{(xi)} & \overline{x^*.y} & = & x^*.\overline{(x.* \mathbin{[\!]} y)} \\
\text{(xii)} & \overline{x^*} & = & x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}
\end{array}$$

In this section, we prove the soundness of these axioms. That is, we prove that each of these axioms is valid.

Theorem B.1. *The axioms defined in this section are sound.*

Proof. We omit the proofs for Axioms (i)–(vi). They follow directly from their definitions in Sections 3 and 4. We also omit the proof for Axiom (viii), which is the much the same as the proof for Axiom (vii).

Proof of (vii): $x.(y \sqcap z) = x.y \sqcap x.z$

From the definitions of . and $\sqcap$, we have to show the following:

$$\{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}} \cap [\![z]\!]^{\mathcal{T}}\} = \\ \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}\} \cap \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\}$$

We rewrite the LHS as follows:

$$\begin{array}{lll} & \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}} \cap [\![z]\!]^{\mathcal{T}}\} & \\ \equiv & \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\} & \text{using (A.3)} \\ \equiv & \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![y]\!]^{\mathcal{T}}\} \cap \{s \frown t \mid s \in [\![x]\!]^{\mathcal{T}} \wedge t \in [\![z]\!]^{\mathcal{T}}\} & \text{using (A.3)} \end{array}$$

This is trivially equivalent to the RHS, so the axiom is valid.

Proof of (ix): $\neg(x\ prefixes\ \overline{y}) \wedge \epsilon \not\leq x \Rightarrow \overline{x.y} = \overline{x.*} \mathbin{[\!]} x.\overline{y}$

For this, we have to prove that $x.y$ is disjoint from $x.*$ and $x.\overline{y}$, and that $\overline{x.*} \mathbin{[\!]} x.\overline{y}$ is the complement of $x.y$.

First, we prove that any trace in $x.y$ is not in $\overline{x.*}$ or $x.\overline{y}$. Take any trace, t, from $x.y$. Because $\epsilon \not\leq x$, the prefix of t must be from x. Therefore, t is not a trace from $\overline{x.*}$. The suffix of t must come from y, so it cannot be in $\overline{y}$.

It is possible, for an expression $x.y$, that t comes from both $x.y$ and $x.\overline{y}$. For example, take the expression $a^+.b$. In this case, the trace $\langle a, a, b\rangle$ is in $a^+.\overline{b}$, because we have the case that $\langle a\rangle$ is in a^+, and $\langle a, b\rangle$ is in $\overline{b}$. However, from the premise $\neg(x\ prefixes\ \overline{y})$, we know that this is not possible for in this case.

Having proved that $x.y$ is disjoint from $x.*$ and $x.\overline{y}$, we also need to prove that $\overline{x.*} \mathbin{[\!]} x.\overline{y}$ is the complement of $x.y$. That is, $x.y \mathbin{[\!]} \overline{x.*} \mathbin{[\!]} x.\overline{y} = *$.

Take any trace t. t either begins with a trace from x, in which case it is in $x.y \mathbin{[\!]} x.\overline{y}$, or it does not, in which case it is $\overline{x.*}$, which includes the empty trace because we know that $\epsilon \not\leq x$. So, we are now left to prove that $x.y \mathbin{[\!]} x.\overline{y} = x.*$. We can rewrite the LHS as follows:

$$\begin{array}{lll} & x.y \mathbin{[\!]} x.\overline{y} & \\ \equiv & x.(y \mathbin{[\!]} \overline{y}) & \text{using the distributivity of . over } [\!] \\ \equiv & x.* & \text{using the complement operator definition} \end{array}$$

This is trivially equivalent to the RHS. Using this, and the proof that $x.y$ is disjoint from $x.*$ and $x.\overline{y}$, we conclude that this axiom is valid.

Proof of (x): $x\ prefixes\ \overline{y} \wedge \epsilon \not\leq x \Rightarrow \overline{x.y} = \overline{x.*} \mathbin{[\!]} x.(\overline{x.* \mathbin{[\!]} y})$

For this, we have to prove that $x.y$ is disjoint from $x.*$ and $x.(\overline{x.* \mathbin{[\!]} y})$, and that $\overline{x.*} \mathbin{[\!]} x.(\overline{x.* \mathbin{[\!]} y})$ is the complement of $x.y$.

First, we prove that any trace in $x.y$ is not in $\overline{x.*}$ or $x.(\overline{x.* \mathbin{[\!]} y})$. Take any trace, t, from $x.y$. Because $\epsilon \not\leq x$, the prefix of t must be from x. Therefore, t is not a trace from $\overline{x.*}$.

However, because $x\ prefixes\ \overline{y}$, t can be a trace in both $x.y$ and $x.\overline{y}$, which is why the rewrite specified by Axiom (ix) is not applicable when $x\ prefixes\ \overline{y}$. As an example, consider the expression $a^+.b$, for which the trace $\langle a, a, b\rangle$, which is in both $a^+.b$ and $a^+.\overline{b}$, because we have the case that $\langle a\rangle$ is in a^+, and $\langle a, b\rangle$ is in $\overline{b}$. This is because the traces in a^+ are prefixes of the traces in $\overline{b}$, so we must discount this possibility. Therefore, the axiom states that the complement trace cannot be in $x.\overline{(x.* \mathbin{[]} y)}$. In the example, this is $a^+.(\overline{a^+.*} \sqcap \overline{b})$ (if Axiom (i) is applied), which specifies that all occurrences of a that occur before another member of Σ^* are "consumed" by the left argument of the . operator. So, any trace that begins with a trace from x, and then ends with a trace not from x or y is not in $x.y$.

Now, we are left to prove that $\overline{x.*} \mathbin{[]} x.\overline{(x.* \mathbin{[]} y)}$ is the complement of $x.y$. That is, $x.y \mathbin{[]} \overline{x.*} \mathbin{[]} x.\overline{(x.* \mathbin{[]} y)} = *$.

Take any trace t. t either begins with a trace from x, in which case it is in $x.y \mathbin{[]} x.\overline{(x.* \mathbin{[]} y)}$, or it does not, in which case it is $\overline{x.*}$, which includes the empty trace because we know that $\epsilon \not\leq x$. So, we are now left to prove that $x.y \mathbin{[]} x.\overline{(x.* \mathbin{[]} y)} = x.*$.

First, we reduce the RHS of the $\mathbin{[]}$ reference to $x.(\overline{x.*} \sqcap \overline{y})$. This expression represents the set of traces that start with a trace from x, and ends with a trace from $\overline{y}$ that does not start with a trace from x. Because $x\ prefixes\ \overline{y} \wedge \epsilon \not\leq x$, we know that x must contain a reference to the iteration operator (or equivalent). We know that $x.y \mathbin{[]} x.\overline{y} = x.*$, however, there is an overlap of traces in these two expressions, as demonstrated above with the expressions $a^+.\overline{b}$. However, for any trace $s^\frown t^\frown u$ such that $s \in [\![x]\!]^{\mathcal{T}}$, $t \in [\![x]\!]^{\mathcal{T}}$, and $t^\frown u \in [\![\overline{y}]\!]^{\mathcal{T}}$, we know that $s^\frown t \in [\![x]\!]^{\mathcal{T}}$ also, because $x\ prefixes\ \overline{y}$. u is then either in y, in which case $s^\frown t^\frown u$ is in $x.y$, or is it in $\overline{y}$, for which we have two cases: either u is also in $\overline{x.*}$, in which case $s^\frown t^\frown u$ is in $x.\overline{(x.* \mathbin{[]} y)}$; or u is in $x.*$, in which case we repeat this process iteratively, until u is not longer in $x.*$. Therefore, $\overline{x.*} \mathbin{[]} x.y \mathbin{[]} x.\overline{(x.* \mathbin{[]} y)} = *$, and we conclude that this axiom is valid.

Proof of (xi): $\overline{x^*.y} = x^*.\overline{(x.* \mathbin{[]} y)}$

For this, we have to prove that $x^*.y$ is disjoint from and the complement of $x^*.\overline{(x.* \mathbin{[]} y)}$.

First, we prove that any trace in $x^*.y$ is not in $x^*.\overline{(x.* \mathbin{[]} y)}$. From the definition of *, we break these two expressions into the following:

$x^+.y \mathbin{[]} y$ and $x^+.\overline{(x.* \mathbin{[]} y)} \mathbin{[]} \overline{(x.* \mathbin{[]} y)}$

which split x^* into the cases in which there are zero and one or more traces coming from x^*.

We now take the expression $x^+.y$ and prove this to be disjoint from the RHS. For any trace, t, in $x^+.y$, t cannot be in $\overline{x.* \mathbin{[]} y}$, as no trace in that expression starts with a trace from x. For any trace, $s^\frown t$, such that $s \in x^+$ and $t \in y$ (using the definition of the . operator), then for $s^\frown t$ to be in $x^+.\overline{(x.* \mathbin{[]} y)}$, it must be that s is from the left expression of . operator (because it cannot be in $x^+.\overline{(x.* \mathbin{[]} y)}$), and y in the right operator, $\overline{(x.* \mathbin{[]} y)}$, which is not possible because any trace in y cannot be in $z \sqcap \overline{y}$ for any trace z, and $\overline{(x.* \mathbin{[]} y)}$ reduces to $\overline{x.*} \sqcap \overline{y}$.

Next, we prove that y is disjoint from the RHS. For any trace, t, in y, t cannot be in $\overline{(x.* \mathbin{[]} y)}$, as proved in the previous paragraph. t is also not in $x^+.\overline{(x.* \mathbin{[]} y)}$, because it can either begin with a trace from x^+ or not. In the case that it begins with a trace from x^+, then it must also by in $x^+.y$, by the fact that adding any number of traces from x^+ on the front of the sequence will still result in a trace from x^+. As already proved, $x^+.y$ is disjoint from $x^+.\overline{(x.* \mathbin{[]} y)}$. In the case that t does not begin with a trace from x, then trivially t is not in $x^+.\overline{(x.* \mathbin{[]} y)}$, Therefore, y is disjoint from $x^+.\overline{(x.* \mathbin{[]} y)} \mathbin{[]} \overline{(x.* \mathbin{[]} y)}$.

Next, we have to prove the $x^*.y \mathbin{[]} x^*.\overline{(x.* \mathbin{[]} y)} = *$. First, we prove that $x^*.y \mathbin{[]} x^*.\overline{y} = *$:

$$\begin{array}{lll} & x^*.y \mathbin{[\!]} x^*.\overline{y} & \\ \equiv & x^*.(y \mathbin{[\!]} \overline{y}) & \text{using distributivity of . over } [\!] \\ \equiv & x^*.* & \text{using the definition of the complement operator} \\ \equiv & (\epsilon \mathbin{[\!]} x.x^*).* & \text{using } x^* = \epsilon \mathbin{[\!]} x.x^* \\ \equiv & \epsilon.* \mathbin{[\!]} x.x^*.* & \text{using distributivity of . over } [\!] \\ \equiv & * \mathbin{[\!]} x.x^*.* & \text{using } x.\epsilon = x = \epsilon.x \\ \equiv & * & \text{using Axiom (iv)} \end{array}$$

We know that $x^*.y \mathbin{[\!]} x^*.\overline{y} = *$, and that because x^* *prefixes* $\overline{y}$, there is an overlap of traces in these two expressions. However, for any trace $s^\frown t^\frown u$ such that $s \in [\![x^*]\!]^{\mathcal{T}}$, $t \in [\![x^*]\!]^{\mathcal{T}}$, and $t^\frown u \in [\![\overline{y}]\!]^{\mathcal{T}}$, we know that $s^\frown t \in [\![x^*]\!]^{\mathcal{T}}$ also, because x^* *prefixes* $\overline{y}$. u is then either in y, in which case $s^\frown t^\frown u$ is in $x.y$, or is it in $\overline{y}$, for which we have two cases: either u is also in $\overline{x.*}$, in which case $s^\frown t^\frown u$ is in $x.\overline{(x.* \mathbin{[\!]} y)}$; or u is in $x.*$, in which case we repeat this process iteratively, until u is not longer in $x.*$. Therefore, $\overline{x.*} \mathbin{[\!]} x^*.y \mathbin{[\!]} x^*.\overline{(x.* \mathbin{[\!]} y)} = *$, and we conclude that this axiom is valid.

Proof of (xii): $\overline{x^*} = x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$

From the definition of *, x^* is reduced to the following:

$$\bigcup_{n \geq 0} [\![x^n]\!]^{\mathcal{T}}$$

Expanding the definition of $\bigcup$, we get the following:

$$[\![x^0]\!]^{\mathcal{T}} \cup [\![x^1]\!]^{\mathcal{T}} \cup [\![x^2]\!]^{\mathcal{T}} \cup [\![x^3]\!]^{\mathcal{T}} \cup \ldots$$

which, from the definition of $[\!]$, is equivalent to the following:

$$x^0 \mathbin{[\!]} x^1 \mathbin{[\!]} x^2 \mathbin{[\!]} x^3 \mathbin{[\!]} \ldots$$

Now, we prove that for every $n \geq 0$, every trace from x^n is not in $\overline{x^*}$. For this, we assume that $\epsilon \not\leq x$, because for any expression of the form x^*, x can be reduced to an expression such that $\epsilon \not\leq x$.

Case $n = 0$: For this case, $x^0 = \epsilon$. The only trace in ϵ is the empty trace, so it is sufficient to prove that $\epsilon \not\leq x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$. For this to be false, it must be the case that $\epsilon \leq x^*$ and $\epsilon \leq \overline{\epsilon \mathbin{[\!]} x.*}$. From the definition of x^*, we know that $\epsilon \leq x^*$, so we need to prove that $\epsilon \not\leq \overline{\epsilon \mathbin{[\!]} x.*}$. Using Theorem B.2(i), we reduce $\overline{\epsilon \mathbin{[\!]} x.*}$ to $\overline{\epsilon} \sqcap \overline{x.*}$. From the definition of $\overline{\epsilon}$ and $\sqcap$, it holds trivially that $\epsilon \not\leq \overline{\epsilon} \sqcap \overline{x.*}$, therefore, $\epsilon \not\leq x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$, and this case holds.

Case $n > 0$: For this case, we prove that for every $n \geq 0$, every trace from x^n is not in $x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$.

First, we consider the case that $x^n \leq x^*$ and that $\epsilon \leq \overline{(\epsilon \mathbin{[\!]} x.*)}$, which would imply that the theorem holds. However, from the case above, $n = 0$, we know that $\epsilon \not\leq \overline{\epsilon \mathbin{[\!]} x.*}$, so this is not the case.

Because we know that $\epsilon \not\leq \overline{(\epsilon \mathbin{[\!]} x.*)}$, there must be some $i < n$ such that $x^i \leq x^*$ and $x^{n-i} \leq \overline{(\epsilon \mathbin{[\!]} x.*)}$. Reducing $\overline{(\epsilon \mathbin{[\!]} x.*)}$ we get $\overline{\epsilon} \sqcap \overline{x.*}$, which represents any non-empty traces that is not in $x.*$. x^{n-i} is clearly not in this set because if we take any trace from x^{n-i}, then either that trace is empty, or it starts with a trace from x. Therefore, it is that case that for $n > 0$, every trace from x^n is not in $x^*.\overline{(\epsilon \mathbin{[\!]} x.*)}$.

Finally, we prove that for every $n \geq 0$, every trace from $\overline{x^*}$ is not in x^n.

Case $n = 0$: For this case, $x^0 = \epsilon$. The only trace in ϵ is the empty trace, so it is sufficient to prove that $\epsilon \not\leq x^*.\overline{(\epsilon \mathbin{[]} x.*)}$, which we proved for the previous case of $n = 0$, so this holds.

Case $n > 0$: For this case, we prove that for every $n > 0$, every trace from $x^*.\overline{(\epsilon \mathbin{[]} x.*)}$ is not in x^n.

Take any trace, t, from $x^*.\overline{(\epsilon \mathbin{[]} x.*)}$. As proved already, $\epsilon \not\leq \overline{(\epsilon \mathbin{[]} x.*)}$, so t must begin with a trace from x^*, followed by a non-empty trace. That non-empty trace must be a trace that does not begin with a trace from $\overline{x.*}$ (recall that $\epsilon \not\leq x$). There does not exist an $n > 0$ such that the suffix of any trace in x^n is a non-empty trace that does not begin with a trace from x. Therefore, we conclude that for every $n > 0$, every trace from $x^*.\overline{(\epsilon \mathbin{[]} x.*)}$ is not in x^n.

Now we have proved that for all $n \geq 0$, x^n is disjoint from $x^*.\overline{(\epsilon \mathbin{[]} x.*)}$. If each of these are disjoint from $x^*.\overline{(\epsilon \mathbin{[]} x.*)}$, then the union of them will also be, so we conclude that this axiom is valid. □

B.2 Additional Theorems

In this section, we present and prove a set of theorems that we use to prove completeness of our axioms, but which are also useful theorems for proving properties about liveness expressions.

Theorem B.2. *The following formulae are valid and are provable using the axioms defined above:*

$$\begin{array}{lrcl}
\text{(i)} & \overline{x \mathbin{[]} y} & = & \overline{x} \sqcap \overline{y} \\
\text{(ii)} & \overline{x \sqcap y} & = & \overline{x} \mathbin{[]} \overline{y} \\
\text{(iii)} & x \mathbin{[]} (y \sqcap z) & = & (x \mathbin{[]} y) \sqcap (x \mathbin{[]} z) \\
\text{(iv)} & x \leq y & \Rightarrow & x \sqcap y = x \\
\text{(v)} & \epsilon \leq x & \Rightarrow & x.* = * = *.x \\
\text{(vi)} & k \geq 1 & \Rightarrow & \overline{x^k} = ([x]^{k-1}).\overline{x.*} \mathbin{[]} x^k.\overline{\epsilon}
\end{array}$$

Proof. (i)–(iv) can be proved using $\overline{\overline{x}} = x$ and the definitions of $[]$ and $\sqcap$, and are also straightforward given the axioms of set theory.

The proof of (v) is proved by expanding the definition of $\leq$ to $\epsilon \mathbin{[]} x = x$, substituting $\epsilon \mathbin{[]} x$ for x, and reducing:

$$\begin{array}{lll}
 & \epsilon \leq x \Rightarrow x.* = * = *.x & \\
\equiv & (\epsilon \mathbin{[]} x).* = * = *.(\epsilon \mathbin{[]} x) & \text{using definition of } \leq \\
\equiv & \epsilon.* \mathbin{[]} x.* = * = *.\epsilon \mathbin{[]} *.x & \text{using distributivity of . over } [] \\
\equiv & * \mathbin{[]} x.* = * = * \mathbin{[]} *.x & \text{using } x.\epsilon = x = \epsilon.x \\
\equiv & * = * = * & \text{Axiom (iv) } (x \mathbin{[]} * = *)
\end{array}$$

The proof of (vi) is less straightforward than these. For this proof, we assume that $\epsilon \not\leq x$ and x is not a iterative definition. We prove the theorem inductively. First, we take the base case of $k = 1$.

Case $k = 1$:

We substitute in 1 for k and simplify:

$$\begin{array}{lll}
 & \overline{x^1} = ([x]^0).\overline{x.*} \mathbin{[]} x^1.\overline{\epsilon} & \\
\equiv & \overline{x} = \epsilon.\overline{x.*} \mathbin{[]} x.\overline{\epsilon} & \text{using definition of } x^k \\
\equiv & \overline{x} = \overline{x.*} \mathbin{[]} x.\overline{\epsilon} & \text{using } x.\epsilon = x
\end{array}$$

From our assumption that $\epsilon \not\leq x$ and x is not iterative, then we know that any trace, t, in $\overline{x}$ either does not start with a trace from x, or it does start with a trace from x, but is then followed by a non-empty trace. So the theorem holds for $k = 1$.

Case $k > 1$:

For this, we assume that the theorem holds for k-1. From the definition of x^k, we expand $\overline{x^k}$ to $\overline{x.x^{k-1}}$, and reduce this.

Additionally, we propose the following lemma:

Lemma B.3. $x.[x]^k = [x]^{k+1} \sqcap \overline{\epsilon}$

Proof. To prove this, we expand $[x]^k$ to the following:

$$\epsilon \;[]\; x \;[]\; x^2 \;[]\; \ldots \;[]\; x^k$$

and are left to prove:

$$x.(\epsilon \;[]\; x \;[]\; x^2 \;[]\; \ldots \;[]\; x^k) = (\epsilon \;[]\; x \;[]\; x^2 \;[]\; \ldots \;[]\; x^{k+1}) \sqcap \overline{\epsilon}$$

Distributing the . over the $[]$ references, and eliminating the choice for ϵ in the RHS (because ϵ cannot be in this expression from the $\sqcap\overline{\epsilon}$ condition), we are left with the trivially true predicate:

$$x \;[]\; x^2 \;[]\; \ldots \;[]\; x^k \;[]\; x^{k+1}) = x \;[]\; x^2 \;[]\; \ldots \;[]\; x^{k+1}$$ □

Returning to the case of $k > 0$, we reduce $\overline{x.x^{k-1}}$ as follows:

$$\begin{array}{lll}
 & \overline{x.x^{k-1}} & \\
\equiv & \overline{x.*} \;[]\; x.\overline{x^{k-1}} & \text{using Axiom (ix)} \\
\equiv & \overline{x.*} \;[]\; x.([x]^{k-2}.\overline{x.*} \;[]\; x^{k-1}.\overline{\epsilon}) & \text{assuming the theorem holds for } k\text{-1} \\
\equiv & \overline{x.*} \;[]\; x.([x]^{k-2}).\overline{x.*} \;[]\; x.x^{k-1}.\overline{\epsilon} & \text{using the distributivity of . over } [] \\
\equiv & \overline{x.*} \;[]\; ([x]^{k-1} \sqcap \overline{\epsilon}).\overline{x.*} \;[]\; x^k.\overline{\epsilon} & \text{using Lemma B.3 and definition of } x^k \\
\equiv & \epsilon.\overline{x.*} \;[]\; ([x]^{k-1} \sqcap \overline{\epsilon}).\overline{x.*} \;[]\; x^k.\overline{\epsilon} & \text{using } x.\epsilon = x = \epsilon.x \\
\equiv & (\epsilon \;[]\; ([x]^{k-1} \sqcap \overline{\epsilon})).\overline{x.*} \;[]\; x^k.\overline{\epsilon} & \text{using the distributivity of . over } [] \\
\equiv & (\epsilon \;[]\; [x]^{k-1}) \sqcap (\epsilon \;[]\; \overline{\epsilon})).\overline{x.*} \;[]\; x^k.\overline{\epsilon} & \text{using (iii)} \\
\equiv & ([x]^{k-1}).\overline{x.*} \;[]\; x^k.\overline{\epsilon} & \text{using } \epsilon \leq [x]^k,\ \epsilon \;[]\; \overline{\epsilon} = *, \text{ and (iv)}
\end{array}$$

Therefore, we conclude that these theorems are valid and can be proved using the axiom system. □

B.3 Completeness

In this section, we present a proof that the axioms defined for the complement operator are complete with respect to normal form. That is, we prove that for any compound expression, x, its complement, $\overline{x}$ can be reduced to normal form using the axioms.

Theorem B.4. *For any liveness expression, $\overline{x}$, such that $x \notin \Sigma$, $x \neq *, \emptyset, \epsilon$, $a.*$, $\overline{x}$ can be further reduced such that it is in normal form.*

Proof. To prove this, we take each possible case for the structure of x separately. We assume that x is reduced to normal form.

Case $x = y \;[]\; z$: Using the induction theorem, we assume that y and z are in normal form. If this is the case, then $y \;[]\; z$ is in normal form.

Case $x = \overline{y \ [] \ z}$: $\overline{y \ [] \ z}$ can be reduced to $\overline{y} \sqcap \overline{z}$ using Theorem B.2(i), and from the induction theorem, the expressions $\overline{y}$ and $\overline{z}$ can be reduced to normal form to propagate complement operators inwards. Should the normal form of $\overline{y}$ or $\overline{z}$ contain any choice operators, these are propagated outwards using Axiom (vi).

Case $x = y \sqcap z$: Using the induction theorem, we know that any complement or sequence operators in y and z are propagated inwards. Any choice operators can be propagated outwards using Axiom (vi).

Case $x = \overline{y \sqcap z}$: $\overline{y \sqcap z}$ can be reduced to $\overline{y} \ [] \ \overline{z}$ using Theorem B.2(ii), and from the induction theorem, the expressions $\overline{y}$ and $\overline{z}$ can be reduced to normal form. If the reduced expressions $\overline{y}$ and $\overline{z}$ are in normal form, then the choice of them is also in normal form.

Case $x = y^*$: With the exception of y being a choice, this expression is already in normal form. In the case that y is a choice, that is, $y = w \ [] \ z$, we can reduce this using Kleene algebra property from Appendix to get $w^*.(z.w^*)^*$.

Case $x = \overline{y^*}$: $\overline{y^*}$ can be reduced to $y^*.\overline{(\epsilon \ [] \ y.*)}$ using Axiom (xii). Using the induction theorem, the expression $\overline{(\epsilon \ [] \ y.*)}$ can be reduced to $\overline{\epsilon} \sqcap \overline{y.*}$. From the induction theorem, $\overline{y.*}$ can be reduced to normal form, while $\overline{\epsilon}$ is in normal form already. Using Axiom (viii), this can be reduced to $(y^*.\overline{\epsilon}) \sqcap (y^*.\overline{y.*})$. If the normal form of $\overline{y.*}$ contains choice or sequence operators, these are propagated appropriately using the axiom system.

Case $x = y.z$: If y and z are in normal form, this expression is in normal form unless y or z contain choice or conjunction operators. If this is the case, propagate the outwards using the distributivity of choice and conjunction over sequential composition.

Case $x = \overline{y.z}$:

If $\epsilon \not\leq x$, then $\overline{y.z}$ can be reduced using either Axiom (ix) or Axiom (x), depending on whether $x \ prefixes \ \overline{y}$ or not. Using the induction theorem, we know that the resulting expressions can be reduced into normal form, therefore, their choice is in normal form.

If $\epsilon \leq x$, then $\overline{y.z}$ can be reduced using Axiom (xi). Even though Axiom (xi) is only applicable to expression of the form $x^*.y$, because x^* is the expression to the left of the sequence operator, we know that $\epsilon \leq y$. In addition, we know that if $\epsilon \leq y$, then by the definition of $prefixes$, it must be the case that $y \ prefixes \ \overline{z}$ (because for every trace, t, in $[[\, z \,]]^{\mathcal{T}}$, there exists an empty trace, s, in $[[\, y^* \,]]^{\mathcal{T}}$ such that $s^\frown t = t$), so it is not possible that that $\neg(y \ prefixes \ \overline{z})$. We are left with the expression $x^*.\overline{(x.* \ [] \ y)}$. Using the same argument for $\overline{(x.* \ [] \ y)}$ as we did for $\overline{(\epsilon \ [] \ y.*)}$ in the proof for the $x = y^*$ case, our expression is in normal form.

The cases $x = y.*$ or $*.y$ are special cases, because in many cases applying the axioms will leave us with the same result; i.e. applying the axioms to $\overline{y.*}$ will result in $\overline{y.*}$. If $y \in \Sigma$, then this is as far as we reduce. If $y = (w \ [] \ z)$, then this should be expanded into $\overline{w.* \ [] \ z.*}$. If $y = w.z$, then $\overline{w.z.*}$ is reduced using the axioms. If $y = z^*$, then $\overline{z^*.*}$ is equivalent to $\overline{*}$, from Theorem B.2(v). The same applies to $*.y$.

We have proved that for each type of liveness expression, x, that satisfies the premise of this theorem, then there exists a series of axioms to reduce $\overline{x}$ into normal form. $\square$