A Verified Algebra for Read-Write Linked Data

Horne, Ross; Sassone, Vladimiro

2014

Horne, R., & Sassone, V. (2013). A verified algebra for read–write Linked Data. Science of Computer Programming, 89, 2-22.

https://hdl.handle.net/10356/80958

https://doi.org/10.1016/j.scico.2013.07.005

A Verified Algebra for Read-Write Linked Data

Ross Horne[a,b], Vladimiro Sassone[a]

[a] *Electronics and Computer Science, University of Southampton, United Kingdom*
[b] *Faculty of Information Technology, Kazakh British Technical University, Almaty, Kazakhstan*

Abstract

The aim of this work is to verify an algebra for high level languages for reading and writing Linked Data. Linked Data is raw data published on the Web and interlinked using a collection of standards. The main innovation is simply to use dereferenceable URIs as global identifiers in data, rather than a key local to a dataset. This introduces significant challenges for managing data that is pulled from distributed sources over the Web. An algebra is an essential contribution to this application domain, for rewriting programs that read and write Linked Data.

To verify the algebra, a syntax, operational semantics and proof technique are introduced. The syntax provides an abstract representation for a high level language that concisely captures queries and updates over Linked Data. The behaviour of the language is defined using a concise operational semantics. The natural notion of behavioural equivalence, contextual equivalence, is shown to coincide with the bisimulation proof technique. Bisimulation is used to verify that the algebra preserves the operational semantics, hence rewrites of programs using the algebra do not change their operational meaning. A novel combination of techniques is used to establish the correctness of the proof technique itself.

Keywords: operational semantics, bisimulation, Linked Data

1. Introduction

This work focuses on high level languages for reading and writing data published on the Web [8]. There is a powerful emerging movement to published raw data on the Web and to interlink the published data by using URIs embedded in the data. Data published in this way is referred to as Linked Data [7, 12]. The Linked Data movement is gaining considerable momentum as major organisations including the UK and US governments [59] and the BBC [32], adopt associated technologies and principles for publishing valuable data.

The "link" in Linked Data refers to the URI and its usage. The URI is a standardised and globally recognised identifier for resources. Instead of publishing documents, as is done for traditional Web sites, raw data is published directly by organisations.

Email addresses: ross.horne@gmail.com (Ross Horne), vs@ecs.soton.ac.uk (Vladimiro Sassone)

URIs are used to identify resources in the published data, thus distinct datasets may unambiguously refer to the same resource. Using URIs in this way is a small conceptual shift. However, this shift enables new opportunities. Since organisations use a single global naming system for identifiers in data, datasets can be interlinked between any organisations. The result is a global dataset, without boundaries or central control, that is constantly changing.

The traditional setting for data is well understood. Without a global naming system such as the URI, each dataset uses its own naming system or key and each dataset is disjoint. Such a system is a closed system, since the boundaries of the data are known; thus classical logic can be used, to determine whether some data does not appear in a dataset for instance. Also, database schemata can be employed to structure the data. Denotational semantics is suited to this setting, where the entire space of the dataset is clearly defined [53].

The Linked Data setting presents significant new challenges. The use of URIs as a global naming system links distributed data sets by allowing them to refer to common resources. Furthermore, a RESTful (HTTP based) protocol [22] can be used to obtain new data from distributed sources based on URIs that appear in known data [21, 39]. An HTTP GET request on a URI can be used to obtain data about the resource the URI represents. URIs that return data in this way are called dereferenceable URIs [33]. In this Linked Data setting, there is no guarantee that dereferencing a URI will find all relevant data about a resource. There may always be relevant data published on the Web that is not obtained by the protocol. Thus, in general, maximal query results cannot be expected, nor can classical logic be employed. Furthermore, due to decentralisation, schemata that constrain data cannot be enforced globally. Operational semantics is suited to this setting, where the full context is not known and operations may not be maximal. Also, operational semantics are suited to specifying how Linked Data evolves, as required to model Read-Write Linked Data [8].

Several technologies for Linked Data standardised by the World Wide Web Consortium (W3C) have found widespread use. These technologies include a light data format called the Resource Description Format (RDF) [40], and the SPARQL query language [30]. RDF is a more loosely structured data format than XML, consisting of RDF triples. An RDF triple resembles the universal idea of a simple sentence in natural language, where a property is used like a verb in natural language to relate a subject and an object. Due to this flexibility, diverse data from different sources can be lifted to RDF. Thus RDF is suited to combining data from many datasets, so that queries can be asked across multiple datasets. Some smart techniques have been employed to implement distributed SPARQL queries [60, 61, 56].

The authors provided the first operational semantics for RDF and SPARQL Query in their FOCLASA workshop paper [37]. The calculus in this work adapts the calculus in the workshop paper so that it also captures SPARQL Update, which became a W3C recommendation in 2013 [25]. SPARQL Update specifies updates over RDF that delete data, insert data and constrain updates according to queries. The first operational semantics for SPARQL Update was also provided by the authors [38]. This work presents a high level language which internalises RDF, SPARQL Query and SPARQL Update with some constructs for concurrency. The update language reuses constructs from the query language, thus it makes sense to consider this more general language

that combines RDF, SPARQL Query and SPARQL Update. In this sense, a new high level domain specific language is proposed. Note that the HTTP operations for harvesting Linked Data are not modelled in this calculus, but they are modelled in related work [21, 39]. However, URIs that appear in the electronic version of this paper are real derefenceable URIs, e.g. *res:Linked_data* .

In this work a natural notion of operational equivalence is used to assign semantics to the terms of a process calculus for Linked Data. An algebra for the calculus is axiomatised using common structures including semirings [28], such that the algebra is sound with respect to the operational equivalences introduced. The equations derived from the algebra can be used to rewrite processes via equational reasoning, which enables quick implementations, essential optimisations and novel programming techniques. Optimisations include finding normal forms for the distribution of updates across collections of datasets, a key problem for Linked Data [31]. Rewriting processes is also relevant to the Big Data application area [1]. For Big Data applications, a process must be rewritten to a form that allows the process to be executed over a fault tolerant, hence scalable, cluster of machines [20].

To be able to express the operational semantics of languages for Linked Data, a novel framework for expressing operational semantics is used. In many ways, the framework goes beyond traditional frameworks for concurrency, such as the π-calculus [50], due to the powerful atomic actions that must be expressed. Indeed the framework raises significant questions about operations for concurrency which are often taken for granted, such as interleaving merge in the presence of synchronisation.

The order of presentation. Sections 2 and 3 explicitly mention the syntactic techniques employed and motivates the applications of an algebra for Linked Data using examples. Readers who prefer to first read the formal definitions can skip straight to Sections 4 and 5. Here the syntax of a high level language and its operational semantics are specified. Section 6 introduces and verifies the proof techniques employed, i.e. the soundness and completeness of bisimulation with respect to contextual equivalence. Finally, Section 7 introduces an algebra over terms in the language and proves the correctness of the algebra with respect to bisimulation.

2. A Bridge Between Operational Semantics and Linked Data

This paper spans several areas of computer science. Therefore it is beneficial to explicitly mention the techniques employed. Techniques drawn from the fields of process calculi and logic are applied to high level programming languages for Linked Data. It is expected that some readers are interested in the implications of this work for Linked Data, while other readers are also interested in the novel framework employed. Both audiences are addressed, which demands a careful exposition of the basic notations and terminology.

All techniques employed here are syntactic. The syntactic approach is suited to the study of programming languages. An abstract syntax of the language is expressed using a BNF grammar [4]. Axioms and rules of the language are then expressed in the style of natural deduction [26]. In a rule several premises, above a horizontal line, lead to a conclusion, below the line. The axioms and rules form a deductive system that defines

relations between terms that define the operational semantics [54] of the language. An operational semantics is a precise executable specification for a programming language. The operational semantics can be implemented using term rewriting logic [48] to yield an interpreter for the language.

Typically operational semantics are expressed according to two approaches. The first approach is to define a reduction system; the second is to define a labelled transition system. A reduction system directly defines the states a process can make a transition to. The process on the left of a reduction relation represents the state before the transition; while the process on the right represents the state after the transition. The labelled transition system goes beyond a reduction system by describing and prescribing a side effect or action for each transition, represented by a label. The action indicates how the process interacts with its context. Actions may be composed with other actions to form larger actions and interactions, thereby providing a composable definition of the operational semantics.

It is conventional for a labelled transition system to be verified by proving that equivalences in the labelled transition system are sound and complete with respect to equivalences in a reduction system [47, 51, 57]. For preliminary versions of this calculus, soundness and completeness have been proven explicitly in this fashion [36, 37]. By proving that an equivalence in the labelled transition is sound and complete, the corresponding bisimulation proof technique can be used in confidence. Bisimulation is easier to use when verifying algebraic properties of a calculus.

The novelty of the approach presented here is that both the reduction system and the labelled transition system are expressed using similar deductive systems. The main distinction between the two systems is that the labelled transition system uses a rule to compose complementary actions, such as stored data and a delete command; whereas the reduction system is not allowed to compose actions, but is instead allowed to use more contextual information than the labelled transition system. Thus the soundness and completeness of the labelled transition system relies on the admissibility of the rule for composing actions. An admissible rule is a concept in proof theory that means that anything that can be derived with the rule can be derived without the rule. Admissibility of a rule is proven by rewriting deduction trees to a form that does not use the rule in question. In proof theory, the admissibility of the cut rule in a logic is the cut elimination theorem for the logic [24, 26, 27]. A paper that explores the intimate connection between the soundness and completeness of an operational semantics and cut elimination in proof theory in detail is future work. This paper focusses on the applications and the intuition that leads to the calculus and techniques presented.

3. Motivating Applications for an Algebra for Linked Data

Before embarking on systematic definitions, motivating examples are provided along with their intuition. The examples introduce RDF [40], SPARQL Query [30] and SPARQL Update [58] as modelled in the calculus introduced in this work. The queries and updates are executed according to their operational semantics. The examples are recast and their implications are discussed.

The reader who prefers to study the formal definitions for syntax and semantics before considering informal motivating examples and discussion, is advised to read

Section 4 ahead of Section 3.

3.1. Some simple sentences expressed as triples

Firstly, consider some real Linked Data taken from DBpedia [13], which lifts data from Wikipedia to RDF. URIs in examples are derefenceable URIs that link to real published Linked Data. The data concerns a retired footballer called Joe Armstrong. Armstrong is assigned a URI *Armstrong* to identify him in Linked Data.

Armstrong *foaf:name* `"Joe Armstrong"`
Armstrong *rdf:type* *Footballer*
Armstrong *dbp:position* *res:Inside_forward*
Armstrong *dbp:birthDate* 1939-01-29
Armstrong *p:clubs* *Gateshead*

The above data consists of RDF triples, where a triple consists of a subject, property and object. The subjects are URIs, *Armstrong* in this case. The properties indicates how the subject is related to the object. Properties are also URIs which are chosen from 'metadata' vocabularies, such as *foaf:name* which is drawn from the popular FOAF vocabulary. The object can be either a URI or a simple term called a literal. The object of the triple with the *dbp:birthDate* property is a date literal; while the football club Gateshead F.C. is identified by a URI.

Note a property *rdf:type* from the core RDF vocabulary is used. This property is used to indicate that Armstrong is a footballer. The class of footballers is identified by a URI, *Footballer*, which is treated as any other URI.

3.2. An atomic commitment for an update

This section considers some updates which modify the example data. The example update demonstrates how a delete, insert and query can be combined such that they occur in an atomic step. To do so, several operators of the calculus are introduced that are described informally.

RDF for the URI *res:Inside_forward* can be obtained from DBpedia. An inside forward was a position in football popular up until the mid 20^{th} century. Instead, modern football teams use attacking midfielders. The process below represents an update which turns an inside forward born before 1950 into an attacking midfielder.

$$\begin{array}{c}(\textit{Armstrong dbp:birthDate } 1939\text{-}01\text{-}29)^{\perp} \| \\ (\textit{Armstrong dbp:position res:Inside\_forward})^{\perp} \| \\ \exists a.\left(\begin{array}{l} \exists x.\left(\begin{array}{l} |(a \textit{ dbp:birthDate } x)| \\ (x \leq 1950\text{-}01\text{-}01) \end{array} \right) \\ (a \textit{ dbp:position res:Inside\_forward}) \\ \tau(a \textit{ dbp:position res:Attacking\_midfielder})^{\perp} \end{array} \right) \end{array}$$

The above sytax represents the initial state of a process containing both the update code, which is the construct beginning $\exists a.(\ldots)$, and the relevant stored triples, indicated by complementation $(\_)^{\perp}$.

In words, the stored triples express *Armstrong*'s date of birth and field position, while the update queries for triples with specific birth dates and playing roles. The query

selects one triple (indicated by $|\_|$) and binds variables to URIs through the existential quantifier, so as to perform the correct update. After the update, the process commits to the state below.

$$\tau\left(\begin{array}{c}(\textit{Armstrong dbp:birthDate } 1939\text{-}01\text{-}29)^{\perp}\| \\ (\textit{Armstrong dbp:position res:Attacking\_midfielder})^{\perp}\end{array}\right)$$

Here τ is a unit time delay to indicate that one atomic step occurs between the two states. Notice that, in the new state, one of the stored triples in the initial state has been deleted and a new stored triple has been inserted.

The update process contains three triples composed together that play different roles. The first triple acts as a query and binds the variables a and x; the second triple has the effect of deleting a stored triple (through interaction with the $(\_)^{\perp}$ construct); the third triple inserts the updated data. Observe that the other component of the update term is a Boolean filter that restricts a's date of birth.

All of these components – the query, delete, insert and filter – are performed in the same atomic step. This ensures that the insert takes place only if the query and filter are satisfied and the delete is successful. A synchronisation operator, the tensor product (formally denoted by $\otimes$, and indicated here simply by juxtaposition), provides the power to compose such atomic actions synchronously. The operational semantics expose some interesting consequences of the tensor product.

3.3. A Quick Application of the Algebra

A basic application for the process algebra is demonstrated here. The update in the previous section can be rewritten to a normal form. The normal form is such that the scope of existential quantifiers is maximal, all deletes are grouped together, followed by all inserts, then all queries, then all filters. According to this, the update in the previous section can be rewritten in the following form.

$$\exists a.\exists x.\left(\begin{array}{l}(a\ \textit{dbp:position res:Inside\_forward}) \\ \tau(a\ \textit{dbp:position res:Attacking\_midfielder})^{\perp} \\ |(a\ \textit{dbp:birthDate}\ x)| \\ (\texttt{year}(x) < 1950)\end{array}\right)$$

This simple rewrite make use of several rules, which shall be introduced formally and proven sound (with respect to bisimulation) in the rest of the paper. Firstly, the scope of the quantifier which binds x can be extended over tensor since x does not appear free in the insert or the delete. Secondly, the tensor operator, which combines the delete, insert, query and constraint, is commutative. Commutativity allows the operations to be reordered.

The above form maps to the concrete syntax for SPARQL Update [58, 25], as follows. Variables are prefixed by `$` in the concrete syntax.

```
DELETE { $a  dbp:position res:Inside_forward }
INSERT { $a  dbp:position res:Attacking_midfielder }
WHERE {
  $a dbp:birthDate $x .
  FILTER (year($x) < 1950)
}
```

This rewrite demonstrates a quick implementation of the update language. Updates in the calculus can be normalised, then rewritten to updates in the concrete syntax. This allows the calculus to be used as a high level language for updating RDF. A more powerful use of the algebra would be inside a query planner that implements SPARQL or another high level language for Linked Data. The algebra can be used to rewrite updates to a form corresponding to an execution plan that suits the underlying architecture [1], while the operational semantics define the execution of the plan.

3.4. A Larger Example Using the Algebra

Now consider a larger example to illustrate more constructs of the calculus. The query below asks for either scientists or footballers with surname Armstrong and a forename beginning with the character `J`. This is expressed with the help of a function `regex` which determines whether a string satisfies a regular expression. The name of the person can be obtained in two ways: either from a single *foaf:name* property or by combining *foaf:givenName* and *foaf:familyName* properties.

$$
\begin{array}{l}
(\textit{Armstrong foaf:name } \texttt{"Joe Armstrong"})^{\perp}\| \\
(\textit{Armstrong rdf:type Footballer})^{\perp}\| \\
\exists a.\left(\begin{array}{l}
\exists z.\left(\begin{array}{l}
\left(\begin{array}{l}
\exists x, y.\left(\begin{array}{l}
|\,(a\ \textit{foaf:givenName}\ x)\,| \\
|\,(a\ \textit{foaf:familyName}\ y)\,| \\
(z = x + \texttt{" "} + y)
\end{array}\right) \oplus |\,(a\ \textit{foaf:name}\ z)\,|
\end{array}\right) \\
\texttt{regex}(z, \texttt{"J.* Armstrong"})
\end{array}\right) \\
\left(\ |\,(a\ \textit{rdf:type Footballer})\,| \oplus |\,(a\ \textit{rdf:type dbp:Scientist})\,|\ \right) \\
\tau P(a)
\end{array}\right)
\end{array}
$$

Here $\oplus$ represents a 'choice' construct and $P(\_)$ is a continuation process which depends on the variable a bound by the query. The configuration above can commit to the configuration below. The two pieces of stored data are used to answer the query, and are then persisted. The URI discovered when answering the query is passed to $P(\_)$.

$$
\tau\left(\begin{array}{l}
(\textit{Armstrong foaf:name } \texttt{"Joe Armstrong"})^{\perp}\| \\
(\textit{Armstrong rdf:type Footballer})^{\perp}\| \\
P(\textit{Armstrong})
\end{array}\right)
$$

The above atomic transition can be derived using the operational semantics. All operators and operational rules are explained in this work.

A disjunctive normal form. Several normal forms can be envisioned for terms of the calculus, depending on the underlying architecture. The query used in the transition

above can be rewritten as the disjunction of four queries in a normal form.

$$\exists a.\left(\begin{array}{l}\exists given, family.\left(\begin{array}{l}|(a\ \mathit{rdf{:}type}\ \mathit{dbp{:}Scientist})|\\ |(a\ \mathit{foaf{:}givenName}\ given)|\\ |(a\ \mathit{foaf{:}familyName}\ family)|\\ \texttt{regex}(given + \texttt{" "} + family, \texttt{"J.* Armstrong"})\end{array}\right)\tau P(a)\end{array}\right)$$
$$\oplus$$
$$\exists a.\left(\begin{array}{l}\exists given, family.\left(\begin{array}{l}|(a\ \mathit{rdf{:}type}\ \mathit{Footballer})|\\ |(a\ \mathit{foaf{:}givenName}\ given)|\\ |(a\ \mathit{foaf{:}familyName}\ family)|\\ \texttt{regex}(given + \texttt{" "} + family, \texttt{"J.* Armstrong"})\end{array}\right)\tau P(a)\end{array}\right)$$
$$\oplus$$
$$\exists a.\left(\exists full.\left(\begin{array}{l}|(a\ \mathit{rdf{:}type}\ \mathit{dbp{:}Scientist})|\\ |(a\ \mathit{foaf{:}name}\ full)|\\ \texttt{regex}(full, \texttt{"J.* Armstrong"})\end{array}\right)\tau P(a)\right)$$
$$\oplus$$
$$\exists a.\left(\exists full.\left(\begin{array}{l}|(a\ \mathit{rdf{:}type}\ \mathit{Footballer})|\\ |(a\ \mathit{foaf{:}name}\ full)|\\ \texttt{regex}(full, \texttt{"J.* Armstrong"})\end{array}\right)\tau P(a)\right)$$

The above rewrite uses many of the algebraic properties of the calculus. It uses the facts that existential quantifiers ($\exists$) and choice operators ($\oplus$) are least upper bounds, that least upper bounds distribute over tensor, that tensor forms a commutative monoid, that the semiring structure of Boolean algebras is a subalgebra of the semiring structure of Kleene algebras as well as algebraic properties of continuations. All of these algebraic properties are proven and discussed in the rest of the paper.

4. An Abstract Syntax for Read-Write Linked Data

The concrete syntax for both RDF and SPARQL Query are specified in W3C recommendations [40, 30]. Here an alternative abstract syntax is used, which is easier to work with for specifying and proving properties of the operational semantics. By using ASCII characters, familiar keywords and abbreviations the official syntax can be recovered [38]. The syntax borrows from both traditional process calculi, such as the π-calculus, and also fragments of linear logic [27]. This work presents a new approach and application for combining process calculi and linear logic [5, 35, 41, 16].

4.1. An Abstract Syntax for RDF Triples

A popular concrete syntax for RDF is N3 [9]. N3 is intended to be simple and intuitive. The main feature is that triples are expressed as URIs in subject, property and object form terminated by a full stop. The full stop is unnecessary in abstract syntax. Thus a triple is expressed as follows.

$$\mathit{Armstrong}\quad \mathit{dbp{:}position}\quad \mathit{res{:}Inside\_forward}$$

Triples are built from URIs and variables, such as *Armstrong* and a. The third term in the triple, the object, may also be a simple data term called a literal. Literals include

$$
\begin{array}{rcl}
A & ::= & (a\ a\ v) \quad \text{triple} \\
& | & A^{\perp} \quad \text{complement} \\
& | & A \otimes A \quad \text{tensor} \\
& | & A \parr A \quad \text{par} \\
& | & \bot \quad \text{nothing} \\
& | & \mathrm{I} \quad \text{true}
\end{array}
$$

$$
\begin{array}{rcl}
\phi & ::= & \mathrm{I} \quad \text{true} \\
& | & 0 \quad \text{false} \\
& | & \phi \oplus \phi \quad \text{or} \\
& | & \phi \otimes \phi \quad \text{and} \\
& | & \neg\phi \quad \text{not} \\
& | & e \quad \text{expression}
\end{array}
$$

x variable $\quad a$ URI or variable

v literal, URI or variable

$$
\begin{array}{rcl}
U & ::= & A \quad \text{action} \\
& | & \phi \quad \text{filter} \\
& | & U \oplus U \quad \text{choice} \\
& | & U \otimes U \quad \text{tensor} \\
& | & U \parr U \quad \text{par} \\
& | & \exists x.U \quad \text{exists} \\
& | & *U \quad \text{iteration} \\
& | & \tau U \quad \text{unit delay} \\
& | & U \| U \quad \text{interleaving merge}
\end{array}
$$

Figure 1: The syntax of filters (ϕ), actions (A) and processes (U, V, P, Q).

decimal numbers 9.9, strings `"SPARQL 1.1 Update"`, and dates 2013-3-21, as defined in the XML Schema Datatype specification [10].

Variables can appear in place of URIs, since triples can be used as patterns in processes where the variables are bound by an existential quantifier. Variables for literals can be distinguished from variables for URIs by a simple type system [37].

This presentation makes several simplifications, in the interest of clarity. A simplification is that '*blank nodes*,' which act as local identifiers in RDF, are not covered due to frequently debated technical difficulties they introduce [44]. In the workshop paper [37], blank nodes are assimilated to fresh name quantification in the π-calculus. Notice also that the use of triples as the basic unit of data could be replaced *mutatis mutandis* by other loosely structured data structures. For instance, quads (viz., an extra URI indicating the provenance of a triple) could be used instead of triples [17, 18].

4.2. A syntax for Boolean filters

Filters are Boolean formulae built from decidable expressions over literals. Typical expressions, e in Fig. 1, include `regex`(x, `J.*`) that evaluates to true if a string substituted for the variable x matches the regular expression `J.*`. Other typical examples include inequality tests, such as $y < 1950$, which are well defined for numbers, dates and strings. Permitted expressions are defined by the SPARQL Query Recommendation [30]. In a query, filters constrain the values that discovered literals can take. Filters embedded in processes allow choices based on a Boolean test, thus *if* branching statements can be encoded.

The style of embedding of Boolean filters in the syntax of updates is similar to the emebedding of Boolean tests in Kleene algebras, as introduced by Kozen [43]. Kozen notes that Kleene algebras share the same semiring structure, characterised by $\otimes$ and $\oplus$. In a Kleene algebra, the semiring structure is typically the concatenation of strings $\otimes$ and choice between strings $\oplus$. In a Boolean algebra, the semiring structure consists of conjunction $\otimes$ and disjunction $\oplus$. The semiring is the common structure, hence the

natural point at which to combine the two structures. Thus in this work, as in the work of Kozen, the same symbol is used for choice and disjunction $\oplus$ and tensor and conjunction $\otimes$. This will not break the semantics since the tensor of two Boolean expressions is the conjunction of the expressions (due to contraction).

As with Kozen's Boolean tests in Kleene algebras, filters are the only terms for which classical negation is meaningful. It is not meaningful to classically negate a triple or process, but it is meaningful to negate an inequality test over numbers. Thus the syntax is carefully designed so that classical negation only applies to filters. (For a similar reasons the syntax only permits linear negation over $\otimes$ and $\parr$, more specifically linear negation is only meaningful for the actions of interleaving processes.)

For all uses of $\otimes$ true, denoted I, is the unit. However, note that I for the Boolean algebra is the top element, but for actions it is not a top element. This is again consistent with Kleene algebras with tests, where the top element of the Boolean algebra coincides with the unit of tensor in the Kleene algebra (and the multiplicative unit in linear logic). False, denoted 0, in the Boolean algebra coincides with the unit of $\oplus$ over processes. These observations, due to Kozen, allow Boolean filters to coexist with actions and processes in a single calculus.

4.3. Connectives for Interaction and Synchronisation

Typically, several triples will be engaged in answering a query or performing an update. Thus the calculus has several primitives for combining triples. These primitives also indicate where interactions between stored data and updates occur.

The tensor product ($\otimes$). The tensor product forces the actions of two processes to occur synchronously in the same atomic step, but without communication. Nothing in either process may interact with anything in the other process. The tensor product is required to express complex queries and updates. Each part of the update happens in the same atomic step using using distinct resources from the environment. Tensor covers the role of a join in relational algebra. The separation of resources suggests a "share nothing" architecture typical of parallel databases and MapReduce clusters [1]. The unit of tensor coincides with I in the Boolean algebra.

Par and complementation ($\parr$ and $^\perp$). Par is similar to tensor, since the actions of the two composed processes occur synchronously. However par forces communication between processes, rather than forbidding it. The complement of a process is the least process that interacts "perfectly" with the original process. For instance, a stored triple interacts perfectly with a command that deletes that triple. Thus stored triples are expressed as complemented triples. Note that par is not regarded as a programming primitive, but as a tool for expressing the operational semantics. The unit of par is $\perp$.

4.4. Operators of the calculus

Processes are combined by several operations which realise the expressive power of the calculus. Par and tensor extend to processes, but complementation does not. The processes to which complementation is restricted form the actions, which are the parts of the processes involved in interactions.

The unit delay prefix (τ). The unit delay prefix indicates that one atomic step must pass before the process is available. In the presence of the tensor product, the unit delay is sufficient to encode temporal ordering. For instance A then B then P can be notated as $A \otimes \tau\,(B \otimes \tau P)$. This works since A is synchronised with the first delay and B is synchronised with the second delay. The unit delay is similar to the next step modality in temporal logic [45, 52].

The existential quantifier ($\exists$). Existential quantifiers explicitly indicate the scope of variables in processes that should be discovered when the process runs. The example binds a variable in a synchronous delete and insert. Thus the subject deleted is the same as the subject inserted.

$$\exists a.\Big((a\ \textit{foaf:name}\ \texttt{"Joe"}) \otimes \tau(a\ \textit{foaf:name}\ \texttt{"Joseph"})^{\perp}\Big)$$

By using an explicit existential quantifier this calculus functions at a higher level of abstraction than SPARQL. At a lower level, a SPARQL "select query" would indicate the a tuple of terms to be returned for each query results. The tuples would then be parsed by some external mechanism then passed to some continuation, say $D(a)$, which displays information known about URI a. If $Q(a)$ is some query involving a, then the process $\exists a.(Q(a) \otimes \tau D(a))$ represents the process which passes the result discovered for a directly to the continuation process after a unit delay.

The choose operator ($\oplus$). The branching operator, choose, allows one of two processes to execute. For queries this has the effect of the UNION operator, which branches a query. This operator is also useful for control flow in processes, since it is also the external choice found in process calculi.

Synchronous iteration ($*$). A requirement of SPARQL is that a query or update could apply more than once. Unbounded iteration is indicated by the Kleene star prefix operator, which allows zero or more copies of a process to synchronously act in one atomic step. Note that iteration differs from replication in common process calculi. In contrast to replication, all iterated copies of a process act simultaneously using disjoint resources.

Explicit iteration allows some powerful updates to be expressed that would otherwise require several separate updates and some low level coding. For instance, the following update mixes iterated and non-iterated parts.

$$\exists x.((\textit{Armstrong dbp:birthDate}\ x) \otimes *\exists a.\exists y.((a\ \textit{dbp:birthDate}\ y) \otimes (y \leq x)))$$

The above process deletes the date of birth of people older than *Armstrong* that are discovered. However, no process can guarantee maximal execution due to the open world assumption for Linked Data; hence some people may be missed.

Interleaving merge with communication. Interleaving merge with communication is different from par and tensor. Interleaving merge is the prevalent parallel composition that appears in the π-calculus [50] and ACP [6], for instance. Either the processes interact, the left process performs an action or the right process performs an action.

4.5. Abbreviations for queries and common operators

Queries appear frequently in examples, however a query is a derived operation in this calculus. A query is expressed as the process that deletes a triple in one step and stores it again in the next step. The triple must exists for the delete to occur, while the insert ensures that the triple persists. Thus given a triple A, a query $|A|$ is represented by the abbreviation $|A| \equiv A \otimes \tau A^{\perp}$.

Note that this implies a linear semantics for queries, in the precise sense that each stored triple can only used once in an atomic operation. The effect is that the triple is locked while it is being used. For example, a triple cannot be used to produce more than one query result, so each query result draws from different data. This abbreviation is useful in the present paper as it simplifies this presentation of the calculus.

The tensor operator is written simply as juxtaposition in the examples, including in the motivating section. Also operators are prioritised, such that unary operators are stronger than binary operators and tensor is stronger than choice, e.g., $(A \otimes B) \oplus C$ is abbreviated $A\,B \oplus C$.

5. An Operational Semantics for Read-Write Linked Data

This section defines an operational semantics for the calculus. Traditionally an operational semantics is expressed using a reduction system and a labelled transition system. In previous work, a reduction system and a labelled transition system are presented for a preliminary version of the calculus [36, 37]. The two systems were found to be intimately connected; thus the presentation here uses two similar systems to express the labelled transition system and reduction system.

5.1. The reductions and labelled transitions explained as commitments

Both the reduction system and the labelled transition system are expressed in terms of a commitment relation. The idea of a commitment relation was first introduced for the polyadic π-calculus by Milner [49]. Milner found that using a commitment relation to express labelled transitions greatly simplified the presentation of the operational semantics and facilitated the definition of rich communication primitives.

A commitment relation indicates how a process can commit to another process that is more deterministic. The commitment relation may commit a process to a state where one action must occur followed by a continuation process. Consider the following example of a choice between two deletes followed by a continuation process.

$$((\textit{Armstrong foaf:member Gateshead}) \oplus (\textit{Armstrong p:clubs Gateshead})) \otimes \tau P$$

According to the rules that are defined and explained in this section, the above process can commit the following process.

$$(\textit{Armstrong p:clubs Gateshead}) \otimes \tau P$$

The second process is more deterministic than the first since one particular triple must be deleted before the process P continues. Conversely, the first process is more non-deterministic, since there is a choice between two deletes before the continuation process proceeds.

Another notation for the above commitment, perhaps more familiar for automata theorists, is the following.

$$\left(\begin{array}{l}(\mathit{Armstrong\ foaf{:}member\ Gateshead}) \oplus \\ (\mathit{Armstrong\ p{:}clubs\ Gateshead})\end{array}\right) \otimes \tau P \xrightarrow{(\mathit{Armstrong\ p:clubs\ Gateshead})} P$$

The action chosen by the commitment (*Armstrong p:clubs Gateshead*) appears as the label above the transition. The continuation process chosen by the commitment P is written on the right of the labelled transition.

In general, a labelled transition, typically written as $P \xrightarrow{A} Q$, indicates that P is a process that can perform the action A to reach state Q. This is the same as stating that P can commit to the process that does the action A and in the next step behaves as Q, which is written $P \rhd A \otimes \tau Q$. The connection between the traditional notation and commitment notation is summarised as follows.

$$\text{Labelled Transitions} \quad P \xrightarrow{A} Q \quad \triangleq \quad P \rhd A \otimes \tau Q \quad \text{“Commit to action } A\text{”}$$

Similarly, reductions over processes are typically expressed by defining a relation between states $P \longrightarrow Q$. The process on the left is the state before the reduction, and the process on the right is the state after one atomic step. Using a commitment relation this is expressed as $P \rhd \tau Q$, since τQ indicates that in the next step process Q is available.

By expressing both reductions and labelled transitions in terms of a commitment relation, this work is able to focus on the essential difference between the systems. The essential difference between the systems is how interactions are modelled. In the rest of this section, reductions are explored first then labelled transitions are investigated.

5.2. The left and right structural rules

The operational semantics relies on two structural congruences, one for each side of the rule. The use of two congruences is similar to the use of two sided sequents in Gallier's presentation of linear logic [23]. It is also reminiscent of the approach of Kobayashi and Yonezawa [41], who also apply fragments of linear logic to operational semantics. The structural congruence applied on the left of a commitment is presented in Fig. 2. The structural congruence fulfils the role of the exchange structural rule from linear logic, which allows formulae to be reorganised.

$$P \parr \bot \equiv P \qquad P \parr Q \equiv Q \parr P \qquad P \parr (Q \parr R) \equiv (P \parr Q) \parr R$$

Figure 2: The structural rules on the left of commitments.

$$P \otimes \mathrm{I} \equiv P \qquad P \otimes Q \equiv Q \otimes P \qquad P \otimes (Q \otimes R) \equiv (P \otimes Q) \otimes R$$
$$\tau\bot \equiv \mathrm{I} \qquad \tau P \otimes \tau Q \equiv \tau\,(P \parallel Q) \qquad A^{\bot\bot} \equiv A \qquad A^{\bot} \parr B^{\bot} \equiv (A \otimes B)^{\bot} \qquad \mathrm{I}^{\bot} \equiv \bot$$

Figure 3: The structural rules on the right of commitments.

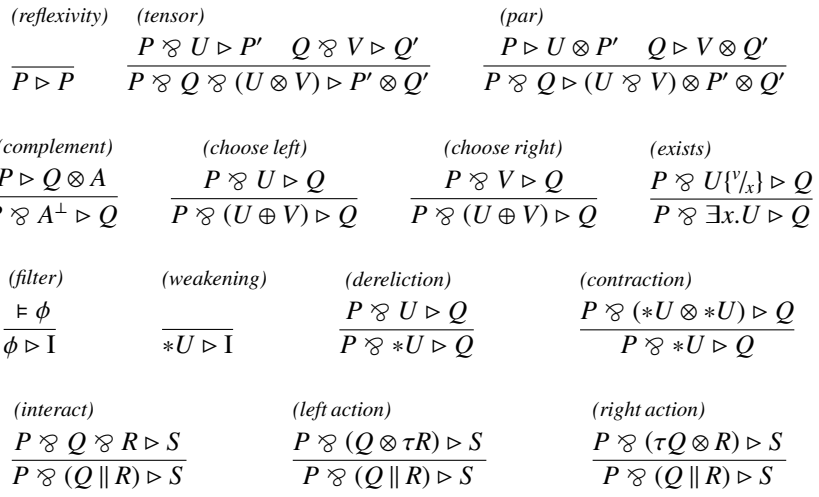

Figure 4: Commitment rules for reductions

The structural congruence on the right of a commitment is given in Fig. 3. On the right, the exchange rule applies to tensor rather than par. Here there are also rules for handling continuations. In particular, the tensor product of two delayed process is equivalent to their parallel composition delayed, an equivalence first derived in [37]. Observe that the standard De Morgan properties of linear logic are also handled by this structural congruence. Including these equivalences as primitive simplifies definitions. Notice however that although $\oplus$ is also commutative, commutativity is not added as a structural rule: the commutativity of plus is instead derived later in Sec. 7 by the observational semantics, in order to obtain a more symmetric reduction system.

5.3. The axioms and rules of the reduction system

The reduction system presents an operational semantics for the calculus. Reductions are derivations with a conclusion which commits the process to the next state. The commitment accounts for interactions between data and updates.

The axioms of the system. The *reflexivity* axiom in Fig. 4 states that a process can always commit to itself. Interactions are captured by combining this axiom with the *complement* rule. Consider the following reasoning.

$$\dfrac{\overline{(\mathit{Armstrong\ p{:}clubs\ Gateshead}) \rhd (\mathit{Armstrong\ p{:}clubs\ Gateshead})}}{(\mathit{Armstrong\ p{:}clubs\ Gateshead}) \parr (\mathit{Armstrong\ p{:}clubs\ Gateshead})^{\perp} \rhd \mathrm{I}}$$

In the above deduction, the axiom introduces the same triple on both sides of the commitment. The complement rule then negates the triple on the right hand side and moves it to the left hand side.

The axiom is also used to introduce delayed processes on the right of a commitment.

The tensor rule. The form of the *tensor* rule, in Fig. 4, is identical to the corresponding rule in linear logic. The two parts of the tensor product are evaluated using separate contexts. In the example below, two stored triples, are simultaneously deleted by two deletes synchronised using the tensor product.

$$\begin{array}{l}(\mathit{Armstrong\ p{:}clubs\ Gateshead})^{\perp} \parr (\mathit{Armstrong\ p{:}clubs\ Leeds})^{\perp} \parr \\ (\mathit{Armstrong\ p{:}clubs\ Gateshead} \otimes \mathit{Armstrong\ p{:}clubs\ Leeds})\end{array} \quad \triangleright \mathrm{I}$$

The tensor rule is essential for this calculus, hence it features in most examples. In examples the symbol for the tensor product is often omitted. Note that the separation of resources enforced by tensor is appropriate to model operations over parallel databases, which are characterised by a 'share nothing' architecture [1].

First order qualification. The *exists* rule in Fig. 4 is essential for accessing URIs and literals in queries. The rule is exactly the rule for first order existential quantification, where some URI or literal is substituted for the variable. A correct choice of substitution allows interactions to take place.

Consider a more substantial example. The process below represents a query which asks for someone associated with the Gateshead Football Club. An existential quantifier binds occurrences of a in subject the triple and the continuation process.

$$\begin{array}{l}(\mathit{Armstrong\ p{:}clubs\ Gateshead})^{\perp} \parr \\ \exists a.\Big(\ |(a\ \mathit{p{:}clubs\ Gateshead})| \otimes \tau P(a) \ \Big)\end{array} \quad \triangleright \tau \left(\begin{array}{l} P(\mathit{Armstrong})\| \\ (\mathit{Armstrong\ p{:}clubs\ Gateshead})^{\perp} \end{array} \right)$$

The above commitment is evaluated by first substituting *Armstrong* for the existentially bound variable. This allows other rules to be applied that interact the query with the data and persist the instantiated continuation process.

The choose rules. The *choose left* and *choose right* rules select one of two branches. The following example deletes a triple relating Armstrong to someone who knows Armstrong or someone who Armstrong knows.

$$\begin{array}{l}(\mathit{Tickell\ foaf{:}knows\ Armstrong})^{\perp} \parr \\ \exists a.(a\ \mathit{foaf{:}knows\ Armstrong} \oplus \mathit{Armstrong\ foaf{:}knows}\ a)\end{array} \quad \triangleright \mathrm{I}$$

In the above example the left branch was chosen. Since the relevant information may be expressed in different ways using triples, the choose rules are essential for offering more than one way for a query or update to interact with data.

Filters. Filters are evaluated by the satisfaction relation $\vDash$, the definition of which is left to the SPARQL recommendation [30]. The relation $\vDash$ is assumed to define a Boolean algebra of filters, consisting of decidable expressions such as matching a string to a regular expression or inequalities over numbers.

The following example embeds a filter in an update. The filter ensures that the

difference between two dates is less than 5 years.

$$
\begin{array}{l}
(\mathit{Armstrong}\ \mathit{dbp{:}birthDate}\ \texttt{"1939-01-29"})^{\perp} \parr \\
(\mathit{Tickell}\ \mathit{dbp{:}birthDate}\ \texttt{"1939-11-15"})^{\perp} \parr \\
(\mathit{Tickell}\ \mathit{p{:}clubs}\ \mathit{Gateshead})^{\perp} \parr \\
\exists a.\left(\begin{array}{l}
|\,(a\ \mathit{p{:}clubs}\ \mathit{Gateshead})\,| \\
\exists x.\left(\begin{array}{l}
|\,(\mathit{Armstrong}\ \mathit{dbp{:}birthDate}\ x)\,| \\
\exists y.\left(\begin{array}{l}
|\,(a\ \mathit{dbp{:}birthDate}\ y)\,| \\
\left(\texttt{abs}(\texttt{year}(x) - \texttt{year}(y)) < 5\right)
\end{array}\right)
\end{array}\right) \\
\tau(\mathit{Armstrong}\ \mathit{foaf{:}knows}\ a)^{\perp}
\end{array}\right)
\end{array}
$$

The rules are applied resulting in x and y being instantiated with dates. The *filter* rule and triples are then evaluated separately, due to the tensor rule. Thus the above process can commit to the state below.

$$
\tau\left(\begin{array}{l}
(\mathit{Tickell}\ \mathit{dbp{:}birthDate}\ \texttt{"1939-11-15"})^{\perp} \parallel (\mathit{Tickell}\ \mathit{p{:}clubs}\ \mathit{Gateshead})^{\perp} \parallel \\
(\mathit{Armstrong}\ \mathit{dbp{:}birthDate}\ \texttt{"1939-01-29"})^{\perp} \parallel (\mathit{Armstrong}\ \mathit{foaf{:}knows}\ \mathit{Tickell})^{\perp}
\end{array}\right)
$$

Synchronous iteration. Notice that all previous examples show updates which are applied only once to data. However, some updates are applied an unbounded number of times to data. Unbounded iteration is explicitly expressed using a Kleene star. The Kleene star is formulated here using *dereliction*, *contraction* and *weakening*. Contraction uses the tensor product to create two synchronous copies of the process, dereliction runs one copy and weakening allows "nothing" to happen.

The following example is prefixed by a Kleene star. There are two triples to which the update applies. Thus contraction is applied once, and dereliction is applied to each branch. This results in the following commitment.

$$
\begin{array}{l}
(\mathit{Meek}\ \mathit{foaf{:}name}\ \texttt{"Joe"})^{\perp} \parr \\
(\mathit{Armstrong}\ \mathit{foaf{:}name}\ \texttt{"Joe"})^{\perp} \parr \\
*\exists a.\left(\begin{array}{l}
(a\ \mathit{foaf{:}name}\ \texttt{"Joe"}) \\
\tau(a\ \mathit{foaf{:}name}\ \texttt{"Joseph"})^{\perp}
\end{array}\right)
\end{array}
\quad \triangleright \tau\left(\begin{array}{l}
(\mathit{Meek}\ \mathit{foaf{:}name}\ \texttt{"Joseph"})^{\perp} \parallel \\
(\mathit{Armstrong}\ \mathit{foaf{:}name}\ \texttt{"Joseph"})^{\perp}
\end{array}\right)
$$

Note that the Kleene star based on a commutative operator, such as tensor, was first investigated by Conway [19].

Interleaving merge with interaction. The standard operational semantics for parallel composition in most common process calculi, such as the π-calculus, is an interleaving semantics. An interleaving semantics has clear operational steps, where an action occurs in each step. It is always possible to tell the order in which actions have happened. Interleaving parallel composition is formalised by an interleaving merge operator $\parallel$ that also permits interactions between processes. There are three rules, the *left* and *right action* rules and the *interaction* rule. The left and right actions rules allow one process to perform an action while another process is delayed. The interact rule synchronises two processes with complementary actions so that they interact. The difference between interleaving merge and par is that with par all processes are involved in an action, but with interleaving merge some processes may not be involved in the current action.

All examples so far in this work hold with respect to both interleaving merge and par, since all resources are used in the example interactions. This is not always the case, since there may be processes and data that are not involved in the current action. The following example exhibits a potential problem when interleaving merge and the tensor product both appear in a calculus.

$$(A \otimes \tau A^{\perp}) \parallel (A \otimes B) \parallel A^{\perp} \parallel B^{\perp} \rhd \tau(A \otimes \tau A^{\perp}) \qquad \text{A valid commitment}$$

$$(A \otimes \tau A^{\perp}) \parallel (A \otimes B) \parallel A^{\perp} \parallel B^{\perp} \not\rhd \tau\,(A \parallel A^{\perp}) \qquad \text{An invalid commitment}$$

The first process exhibits a valid commitment involving the tensor and interact rules. The first process is delayed, and the two components of the second process interact with the stored data. In contrast, the second pair of process should not form a valid commitment. To produce the invalid commitment, B is consumed but the wrong A is consumed. However, the invalid commitment would be a derivable if $\invamp$ and $\parallel$ were collapsed to a single operation. This subtlety forces par and interleaving merge to be distinguished in an interleaving semantics for parallel composition.

Note that interleaving semantics are perhaps the most widely understood semantics for process calculi. Due to the expressive power of the calculus, other interleaving process calculi such as π-calculus (without fresh names) can be embedded in this framework [36]. However, there do exist alternative semantics for process calculi such as non-interleaving semantics [18]. The operators in a non-interleaving semantics may be simpler since par and merge do not need to be distinguished, and also the units I, $\perp$ and τ can be identified as a single unit [29]. However, non-interleaving semantics for expressive calculi are not widely understood. Future work is to develop a non-interleaving operational semantics sufficiently expressive for modelling rich interactions with data.

5.4. A Labelled Transition System

The operational semantics can be expressed as a labelled transition system. This provides an alternative operational semantics to the reduction system, where processes can be evaluated without the entire context. An action keeps track of the interface between the process and the context.

The rules for the labelled transition system are presented in Fig. 5. Most of the rules are special cases of the rules in the reduction system. The special case is obtained by taking the context in a reduction rule to be the process $\perp$. Thus an explanation of each individual rule is not repeated. Instead an explanation and examples of how actions are derived is elaborated in this section. Particular attention is paid to the the *cut* rule, which does not appear in the commitment relation for the reduction system.

5.5. The interaction of a stored triple and a delete

The *reflexivity* axiom is used to initiate actions. The following commitment indicates that the stored triple can interact with a context that uses the stored triple.

$$(\mathit{Armstrong\ foaf{:}knows\ Tickell})^{\perp} \rhd (\mathit{Armstrong\ foaf{:}knows\ Tickell})^{\perp}$$

$$\textit{(cut)}\ \frac{U \rhd A \otimes U' \quad V \rhd A^{\perp} \otimes V'}{U \parr V \rhd U' \otimes V'} \qquad \textit{(tensor)}\ \frac{U \rhd P \quad V \rhd Q}{U \otimes V \rhd P \otimes Q} \qquad \textit{(par)}\ \frac{U \rhd A \otimes P \quad V \rhd B \otimes Q}{U \parr V \rhd (A \parr B) \otimes P \otimes Q}$$

$$\textit{(reflexivity)}\ \frac{}{P \rhd P} \qquad \textit{(filter)}\ \frac{\vDash \phi}{\phi \rhd \mathrm{I}} \qquad \textit{(exists)}\ \frac{U\{^{v}/_{x}\} \rhd Q}{\exists x.U \rhd Q} \qquad \textit{(weakening)}\ \frac{}{*U \rhd \mathrm{I}} \qquad \textit{(dereliction)}\ \frac{U \rhd Q}{*U \rhd Q} \qquad \textit{(contraction)}\ \frac{*U \otimes *U \rhd Q}{*U \rhd Q}$$

$$\textit{(choose left)}\ \frac{U \rhd Q}{U \oplus V \rhd Q} \qquad \textit{(choose right)}\ \frac{V \rhd Q}{U \oplus V \rhd Q} \qquad \textit{(interact)}\ \frac{Q \parr R \rhd S}{Q \parallel R \rhd S} \qquad \textit{(left action)}\ \frac{Q \otimes \tau R \rhd S}{Q \parallel R \rhd S} \qquad \textit{(right action)}\ \frac{\tau Q \otimes R \rhd S}{Q \parallel R \rhd S}$$

Figure 5: Commitment rules for labelled transitions

For the following process, more than one possible action can appear on the right of the commitment. The action that appears corresponds to the contexts the process may interact with.

$$\exists a. \left(\begin{array}{l} (a\ \textit{foaf:knows}\ \textit{Armstrong}) \\ \oplus \\ (\textit{Armstrong}\ \textit{foaf:knows}\ a) \end{array} \right) \rhd (\textit{Armstrong}\ \textit{foaf:knows}\ \textit{Tickell})$$

The two processes above emit complementary actions. Hence the actions may interact by using the cut rule. The result of the interaction is the same commitment that was derived using the rules of the reduction system in Sec. 5.3.

5.6. The general case for two party interaction

The previous example demonstrated an interaction between two parties — a delete and a stored triple. Now consider the general case of two party interactions. Two processes can interact with each other if they offer complementary information on the label. Complementary information is expressed using a complementation operator, thus the complement of a label A is $A^{\perp}$.

The following derivation is obtained by applying the cut rule then interact rule from Fig. 5. The combined rule is reminiscent of interaction between an action and its co-action in CCS. The CCS rule is shown on the right for comparison.

$$\frac{\dfrac{P \rhd A \otimes \tau P' \quad Q \rhd A^{\perp} \otimes \tau Q'}{P \parr Q \rhd \tau P' \otimes \tau Q'}}{P \parallel Q \rhd \tau\,(P' \parallel Q')} \qquad\qquad \frac{P \xrightarrow{A} P' \quad Q \xrightarrow{\overline{A}} Q'}{P \parallel Q \xrightarrow{\tau} (P' \parallel Q')}$$

Notice that both $\parr$ and $\parallel$ are used when composing processes in parallel that interact. The former composition synchronises the first actions of each process and allows interaction, while the latter is an interleaving merge operator. A similar technique is used by Bergstra in ACP [6], where the interleaving merge operator is decomposed into a left action, a right action, and synchronisation with communication. In this work, the latter role of synchronisation with communication is taken by par, which is similar to

the homonymous operator from linear logic. The general case of synchronisation with communication between two parties is presented above. The cases of the left and right action are defined by the left action and right action rules in Fig. 5.

Contrarily to labelled transitions, a reduction system requires all information to perform a reduction. To do so, its rules carry the the whole context required to derive an interaction, by using par. Thus the reduction system defined previously deals with interactions similarly to proofs in cut free linear logic.

No claim is made anywhere in this work that the commitment relation is an encoding of an operational semantics in linear logic. Specifically, the syntax of actions only are in the syntax of multiplicative linear logic. Thus actions form a controlled subsystem within processes, where the power of linear logic to control interactions is exploited. Complementation identifies co-actions that may interact with actions, as in the above illustration. Furthermore, the tensor product forbids interactions whilst par permits interactions. This use of par and tensor enables the more general multi-party scenarios, where more than one piece of data is consumed. The approach of using multiplicative linear logic as actions is related to higher dimensional automata [55] and proved transitions [14]. However, the usage here should not be confused with the intuitionistic linear logic approach to behavioural types [2, 16].

5.7. Multi-party interactions using the labelled transition system

For expressive queries and updates, more than one triple may be involved in an interaction. The following process involves a query consisting of two triples followed by a continuation. Both the queries contribute to the label, while all parts contribute to the continuation process. Thus the following commitment represents a labelled transition.

$$\exists a.\left(\begin{array}{l} |(a\ \textit{rdf:type Footballer})| \\ |(a\ \textit{p:clubs Gateshead})| \\ \tau P(a) \end{array}\right) \triangleright \begin{array}{l} (\textit{Armstrong rdf:type Footballer}) \\ (\textit{Armstrong p:clubs Gateshead}) \\ \tau\left(\begin{array}{l} (\textit{Armstrong rdf:type Footballer})^{\perp}\| \\ (\textit{Armstrong p:clubs Gateshead})^{\perp}\| \\ P(\textit{Armstrong}) \end{array}\right) \end{array}$$

The commitment below represents a labelled transition for two stored triples composed in parallel. Firstly the interleaving merge is converted to par. Then axioms are combined using the par rule.

$$\begin{array}{l} (\textit{Armstrong rdf:type Footballer})^{\perp}\| \\ (\textit{Armstrong p:clubs Gateshead})^{\perp} \end{array} \triangleright \left(\begin{array}{l} (\textit{Armstrong rdf:type Footballer}) \\ (\textit{Armstrong p:clubs Gateshead}) \end{array}\right)^{\perp}$$

The De Morgan properties align the two actions, so that it is clear that they are complementary. Therefore the two processes may interact using the cut rule. The cut rule retains the continuations of the first process, but cancels out the actions. This results in the following commitment.

$$\begin{array}{l} (\textit{Armstrong rdf:type Footballer})^{\perp}\| \\ (\textit{Armstrong p:clubs Gateshead})^{\perp}\| \\ \exists a.\left(\begin{array}{l} |(a\ \textit{rdf:type Footballer})| \\ |(a\ \textit{p:clubs Gateshead})| \\ \tau P(a) \end{array}\right) \end{array} \triangleright \tau\left(\begin{array}{l} (\textit{Armstrong rdf:type Footballer})^{\perp}\| \\ (\textit{Armstrong p:clubs Gateshead})^{\perp}\| \\ P(\textit{Armstrong}) \end{array}\right)$$

To derive the above commitment in another way, cut can be applied twice such that each application of cut matches one stored triple with part of the update. Both approaches to deriving the above commitment are equivalent. This equivalence of derivations is used as the principal case in the proof of the results, provided in the next section.

6. Soundness and Completeness of the Operational Semantics

Given a labelled transition system and a reduction system, the former can be evaluated for soundness and completeness against the latter. This is done by proving that the natural notion of equivalence over the labelled transition system coincides with the natural notion of equivalence for the reduction system. These notions are bisimulation and contextual equivalence, respectively.

Bisimulation is a proof technique to verify algebraic properties of a calculus. It it essential to have soundness of bisimulation. Soundness ensures that any algebraic property established using bisimulation also holds in the reduction system. Completeness shows that the bisimulation proof technique can be used in every case to establish a property.

In this presentation of the operational semantics the reduction system and the labelled transition system are intimately connected. The intermediate results highlight this connection, through a theorem that proves the admissibility of the cut rule in Sec. 6.2 and the congruence theorem in Sec. 6.3.

6.1. Equivalences on Processes

There are two natural notions of operational equivalence, respectively for reduction systems and for labelled transitions. Both are defined as relations over processes. The natural property which an operational equivalence should satisfy is reduction closure. Reduction closure ensures that if one process can perform a reduction, then an equivalent process can also perform a reduction to an equivalent state. This is a coinductive principle, which is defined as follows.

Definition 1 (Reduction closure)**.** *A reduction closed relation $\mathcal{R}$ is such that if $P\ \mathcal{R}\ Q$ and $P \rhd \tau P'$ then there exists some Q' such that $Q \rhd \tau Q'$ and $P'\ \mathcal{R}\ Q'$.*

Reduction closure by itself is not sufficient to generate a meaningful equivalence. For example any two stored triples are equivalent since neither triple can perform a silent action. Context can of course have a drastic effect on the behaviour of a process. Therefore an equivalence should also satisfy context closure defined as follows, where contexts are defined as usual to be terms with a hole.

Definition 2 (Context closure)**.** *A context close relation $\mathcal{R}$ is such that $P\ \mathcal{R}\ Q$ yields $CP\ \mathcal{R}\ CQ$, for all contexts C.*

Operational equivalence accounts for both reduction closure and context closure by the same relation. By ensuring that the relation is symmetric, an equivalence over processes is defined.

Definition 3 (Contextual equivalence)**.** *Contextual equivalence, written $\simeq$, is the greatest symmetric, reduction closed, context closed relation.*

Contextual equivalence can be difficult to work with directly. Context closure at every step means that there are many cases to check, potentially infinitely many.

An alternative notion of bisimulation is defined using labelled transitions. Bisimulation, like reduction closure, has a coinductive definition, except that the labels account for the context in which a transition takes place. The notion of bisimulation is defined as follows. Here strong bisimulation is used, which accounts for every operational step.

Definition 4 (Bisimulation)**.** *A bisimulation $\mathcal{R}$ is a symmetric relation such that, for any label A, if $P \mathcal{R} Q$ and $P \rhd A \otimes \tau P'$ then there exists some Q' such that $Q \rhd A \otimes \tau Q'$ and $P' \mathcal{R} Q'$.*

Bisimilarity is then defined as the greatest bisimulation, which implies that every bisimulation is contained in bisimilarity. Thus, to establish that two processes are bisimilar it is sufficient to show that there exists a bisimulation which relates them.

Definition 5 (Bisimilarity)**.** *Bisimilarity, written $\sim$, is the greatest bisimulation.*

It is immediate that bisimulations and contextual equivalences are reflexive and transitive relations, hence are equivalence relations. The definition of bisimulation must be verified to be correct with respect to contextual equivalence for this calculus, as done for other calculi [15, 46, 57]. Correctness of bisimulation is proven by the results in this section.

6.2. The Admissibility of the Cut Rule

The main difference between the reduction system and the labelled transition system is that the reduction system does not employ the *cut* rule in Fig. 5. The two systems should therefore be compared by proving that reductions derived using the cut rule can always be derived without using the cut rule, i.e. the cut rule is admissible.

The cut rule is restricted to allowing only actions to interact. Despite this restriction, the proof of the admissibility of the cut rule resembles cut elimination in proof theory. Since the calculus is not symmetric with respect to negation, most cases in the proof are the simple commutative cases. However, the principal case for par and tensor is harder and constitute the point of main interest in the proof.

Theorem 1 (Admissibility of cut)**.** *A commitment $P \rhd Q$ holds in the reduction system if and only if it holds in the labelled transition system.*

Proof. A cut involving the reflexivity axiom can be eliminated. The following two proof trees are equivalent.

$$\dfrac{P \rhd A^{\perp} \otimes Q \quad \overline{A \rhd A}\ \textit{reflexivity}}{P \parr A \rhd Q}\ \textit{cut} \qquad \text{iff} \qquad \dfrac{P \rhd A^{\perp} \otimes Q}{P \parr A \rhd Q}\ \textit{complement}$$

Thus cuts involving reflexivity can be eliminated. This is the basis of an inductive proof of the admissibility of the cut rule, that proceeds by showing that in each case the application of the cut rule can be pushed up towards the leaves of the proof tree.

Commutative cases push the cut rule up the derivation tree past a rule that does not take part in the interaction. All commutative cases are easy and have a similar form. Only one commutative case for interleaving merge is shown here.

Consider the commutative case for interleaving merge, where an interaction takes place. The following two proof trees are interchangeable.

$$\dfrac{\dfrac{P \parr Q \rhd A \otimes P'}{P \parallel Q \rhd A \otimes P'}\ \mathit{interact} \qquad R \rhd A^{\perp} \otimes R'}{(P \parallel Q) \parr R \rhd P' \otimes R'}\ \mathit{cut} \quad \text{iff} \quad \dfrac{P \parr Q \rhd A \otimes P' \qquad R \rhd A^{\perp} \otimes R'}{\dfrac{P \parr Q \parr R \rhd P' \otimes R'}{(P \parallel Q) \parr R \rhd P' \otimes R'}\ \mathit{interact}}\ \mathit{cut}$$

Rules above the cut use the labelled transition system and rules below the cut use the reduction system.

Two cuts in succession can be rewritten to a single cut. Consider two cuts applied in succession as follows.

$$\dfrac{\dfrac{P \rhd A \otimes B \otimes P' \qquad U \rhd A^{\perp} \otimes Q}{P \parr U \rhd B \otimes P' \otimes Q}\ \mathit{cut} \qquad V \rhd B^{\perp} \otimes R}{P \parr U \parr V \rhd P' \otimes Q \otimes R}\ \mathit{cut}$$

The successive cuts can be combined using the par rule. This results in the following proof tree.

$$\dfrac{P \rhd A \otimes B \otimes P' \qquad \dfrac{U \rhd A^{\perp} \otimes Q \qquad V \rhd B^{\perp} \otimes R}{U \parr V \rhd (A^{\perp} \parr B^{\perp}) \otimes Q \otimes R}\ \mathit{par}}{P \parr U \parr V \rhd P' \otimes Q \otimes R}\ \mathit{cut}$$

The De Morgan's property $A^{\perp} \parr B^{\perp} \equiv (A \otimes B)^{\perp}$ enables the new cut.

Since the tensor and par are the only connective involved in interactions, there is one principal case. The following proof tree represents this scenario.

$$\dfrac{\dfrac{U_0 \rhd A \otimes P \qquad U_1 \rhd B \otimes Q}{U_0 \otimes U_1 \rhd A \otimes B \otimes P \otimes Q}\ \mathit{tensor} \qquad \dfrac{V_0 \rhd A^{\perp} \otimes R \qquad V_1 \rhd B^{\perp} \otimes S}{V_0 \parr V_1 \rhd (A^{\perp} \parr B^{\perp}) \otimes R \otimes S}\ \mathit{par}}{(U_0 \otimes U_1) \parr V_0 \parr V_1 \rhd P \otimes Q \otimes R \otimes S}\ \mathit{cut}$$

The cut can be broken into two separate cuts, each of which can be eliminated. This results in the following proof tree.

$$\dfrac{\dfrac{U_0 \rhd A \otimes P \qquad V_0 \rhd A^{\perp} \otimes R}{U_0 \otimes V_0 \rhd P \otimes R}\ \mathit{cut} \qquad \dfrac{U_1 \rhd B \otimes Q \qquad V_1 \rhd B^{\perp} \otimes S}{U_1 \parr V_1 \rhd Q \otimes S}\ \mathit{cut}}{(U_0 \otimes U_1) \parr V_0 \parr V_1 \rhd P \otimes Q \otimes R \otimes S}\ \mathit{tensor}$$

The rule below the cuts is a rule from the reduction system.

All cases are covered hence, by induction, the cut rule is admissible. □

The immediate corollary of the above result is that a bisimulation is a reduction closed relation. Also, reversing the elimination process turns a reduction into a labelled transition. Both corollaries are used in the proof of correctness of bisimilarity.

6.3. Labels Respect the Context

Algebraic rules should be applicable in any context. Although reduction closure by itself does not give context closed relations, bisimulation does. This is proved by the following result. The interesting case is the Kleene star, which mixes inductive and coinductive reasoning.

Theorem 2. *Bisimilarity is context closed.*

Proof. All cases are proven by assuming the existence of a bisimulation, and then demonstrating that a bisimulation which contains the context in question can be constructed, for each context.

Consider the case of delay. Let $\sim_0$ be a bisimulation and define $\sim_1$ to be the least symmetric relation including $\sim_0$ such that if $P \sim_0 Q$ then $\tau P \sim_1 \tau Q$. Assume that $P \sim_0 Q$. By the reflexivity axiom, if $\tau P \rhd \tau P$ then $\tau Q \rhd \tau Q$ and $P \sim_1 Q$. Hence $\sim_1$ is a bisimulation in this case.

Consider the case of choice. Assume that there exists a bisimulation $\sim_0$. Let $\sim_1$ be the least equivalence relation extending $\sim_0$ such that if $U \sim_0 V$ then $U \oplus W \sim_1 V \oplus W$. Assume that $U \sim_0 V$ and suppose that that, the first transition below holds. Since $U \sim_0 V$, if $U \rhd A \otimes \tau P$ then there exists some Q such that $V \rhd A \otimes \tau Q$, such that $P \sim_0 Q$. So the second transition below holds and $P \sim_1 Q$, as required.

$$\frac{U \rhd A \otimes \tau P}{U \oplus W \rhd A \otimes \tau P} \quad \text{yields} \quad \frac{V \rhd A \otimes \tau Q}{V \oplus W \rhd A \otimes \tau Q}$$

Hence $\sim_1$ is a bisimulation. The case for exists is similar.

Consider the case of interleaving. Suppose that $\sim_0$ is a bisimulation and let $\sim_1$ be the least equivalence extending $\sim_0$ such that if $P \sim_0 Q$ then $P \parallel R \sim_1 Q \parallel R$. There are four cases to check. Only the case for cut over par is verified here.

Assume that $P \sim_0 Q$ hence $P \parallel R \sim_1 Q \parallel R$. Also assume, $P \rhd A \otimes B \otimes \tau P'$, hence there exists Q' such that $Q \rhd A \otimes B \otimes \tau Q'$ and $P' \sim_0 Q'$. Thus the transition on the left yields the transition on the right.

$$\frac{\dfrac{P \rhd A \otimes B \otimes \tau P' \quad R \rhd A^{\perp} \otimes C \otimes \tau R'}{P \parrow R \rhd B \otimes C \otimes \tau (P' \parallel R')}}{P \parallel R \rhd B \otimes C \otimes \tau (P' \parallel R')} \quad \text{yields} \quad \frac{\dfrac{Q \rhd A \otimes B \otimes \tau Q' \quad R \rhd A^{\perp} \otimes C \otimes \tau R'}{Q \parrow R \rhd B \otimes C \otimes \tau (Q' \parallel R')}}{Q \parallel R \rhd B \otimes C \otimes \tau (Q' \parallel R')}$$

Furthermore, $P' \parallel R \sim_1 Q' \parallel R$, as required. All four cases result in an interleaving parallel composition continuation in this form. Hence $\sim_1$ is a bisimulation in this case.

The cases of tensor and par are similar to interleaving. Given a bisimulation $\sim_0$, define $\sim_1$ to be the least symmetric relation extending $\sim_0$ such that if $P \sim_0 Q$ then $P \parallel R \sim_1 Q \parallel R$ and either $P \otimes R \sim_1 Q \otimes R$ or $P \parrow R \sim_1 Q \parrow R$. There are one and two cases to check respectively.

Consider the case of the Kleene star. Assume that $\sim_0$ is a bisimulation. Let $\sim_1$ be the least equivalence extending $\sim_0$, such that if $U \sim_0 V$ then $*U \sim_1 *V$, and, recursively, if both $P_0 \sim_1 Q_0$ and $P_1 \sim_1 Q_1$ then $P_0 \parallel P_1 \sim_1 Q_0 \parallel Q_1$. Assume $U \sim_0 V$, hence $*U \sim_1 *V$. There are three cases to consider for weakening, dereliction and contraction.

The case of the weakening rule is trivial. If $*U \rhd \mathrm{I}$ then $*V \rhd \mathrm{I}$ and $\mathrm{I} \sim_1 \mathrm{I}$.

For dereliction, suppose the first transition below holds. Since $U \sim_0 V$ and $U \rhd A \otimes \tau P$, there exists a Q such that $V \rhd A \otimes \tau Q$ and $P \sim_0 Q$. Hence the second transition below holds and $P \sim_1 Q$.

$$\frac{U \rhd A \otimes \tau P}{*U \rhd A \otimes \tau P} \quad \text{yields} \quad \frac{V \rhd A \otimes \tau Q}{*V \rhd A \otimes \tau Q}$$

For contraction, proceed by induction on the derivation of a transition. Suppose that the first transition below holds. By induction, since $*U \rhd A \otimes \tau P_0$, there exist Q_0 such

$*V \rhd A \otimes \tau Q_0$ and $P_0 \sim_1 Q_0$. Similarly, since $*U \rhd B \otimes \tau P_1$, there exist Q_1 such $*V \rhd B \otimes \tau Q_1$ and $P_1 \sim_1 Q_1$. Hence the second transition below holds and $P_0 \parallel P_1 \sim_1 Q_0 \parallel Q_1$.

$$\frac{*S \rhd A \otimes \tau P_0 \quad *S \rhd A \otimes \tau P_1}{*S \otimes *S \rhd A \otimes B \otimes \tau (P_0 \parallel P_1)} \quad \text{yields} \quad \frac{*S \rhd A \otimes \tau Q_0 \quad *S \rhd A \otimes \tau Q_1}{*S \otimes *S \rhd A \otimes B \otimes \tau (Q_0 \parallel Q_1)}$$
$$*S \rhd A \otimes B \otimes \tau (P_0 \parallel P_1) \qquad\qquad *S \rhd A \otimes B \otimes \tau (Q_0 \parallel Q_1)$$

Hence by induction over the derivation of a transition, $\sim_1$ is a bisimulation.

This covers all cases, hence bisimulation is closed under all contexts. □

Context closure is essential to establishing soundness of bisimilarity.

6.4. Soundness and Completeness of Bisimulation

Here it is confirmed that bisimulation is sound and complete with respect to contextual equivalence. Soundness is essential since algebraic properties proven using bisimulation also hold for contextual equivalence. Completeness ensures that all equivalences can be proven using bisimulation.

Soundness of bisimulation is established by proving that bisimulation is a context equivalence. The proof of soundness uses both intermediate theorems in the previous sections.

Theorem 3 (Soundness of bisimulation)**.** *If $P \sim Q$ then $P \simeq Q$.*

Proof. Reduction closure follows from Theorem 1, while context closure follows from Theorem 2. □

As expected for coinductive techniques, the proof of completeness is easier than soundness. Completeness is proven by demonstrating that contextual equivalence is a bisimulation. The proof follows by finding a suitable context for each label. Completeness is made even easier due to the adjunction which defines linear negation.

Theorem 4 (Completeness of bisimulation)**.** *If $P \simeq Q$ then $P \sim Q$.*

Proof. Suppose that $P \simeq Q$. By context closure $P \parr A^\perp \simeq Q \parr A^\perp$; and by reduction closure if $P \parr A^\perp \rhd \tau P'$, then there exists a process Q' such that $Q \parr A^\perp \rhd \tau Q'$ and $P' \simeq Q'$.

A simple induction ensures that $P \parr A^\perp \rhd \tau P'$ if and only if $P \rhd A \otimes \tau P'$ in the reduction system, so, by Theorem 1, $P \rhd A \otimes \tau P'$ holds in the labelled transition system. Thus if $P \rhd A \otimes \tau P'$ then Q', chosen above, is such that $Q \rhd A \otimes \tau Q'$. Hence contextual equivalence is a bisimulation. □

Thus the bisimulation proof technique is sound and complete with respect to the natural notion of operational equivalence. Therefore bisimulation can be used in confidence in the next section.

7. An Algebra for Rewriting Updates

There are several reasons why an algebra for processes is desirable. Equivalence checking is useful to programmers, who need confirmation that writing a process in different ways has the same meaning. An algebra is also important to compiler engineers, who optimise implementations of languages based on the calculus. Two programs may have the same operational behaviour, but may differ in efficiency when deployed on a specific computer architecture.

Query planners make use of an algebra, referred to as relational algebra in relational databases. The algebra is used to rewrite a query so that it can be executed as efficiently as possible. The algebra verified in this section applies to all processes; hence the techniques used for query planning can be applied to more general processes. Furthermore, the algebra employed is proven to be correct using the bisimulation proof technique.

Further to enhancing programming techniques and implementations, an algebra provides objective justification for the calculus. If an operator satisfies well understood algebraic properties, then the operator is more likely to be correct. For instance idempotent semirings, which are common structures in a wide range of applications in computer science, appear in this calculus. Furthermore, since structures such as semirings are well understood their properties may be exploited.

7.1. Algebraic properties of processes

The algebra of the calculus combines several well known structures. This section summarises the familiar structures, and defines and an algebra for the entire calculus.

Commutative idempotent semirings are ubiquitous structures in computer science. Idempotent semirings are used in concurrent constraint programming [11], which shares similar aims to the approach explored here.

Definition 6 (Idempotent semiring)**.** *A semiring consists of a monoid* $(U, \otimes, \mathrm{I})$ *and a commutative monoid* $(U, \oplus, 0)$*, where tensor distributes over choice,* $(U \oplus V) \otimes W = (U \otimes W) \oplus (V \otimes W)$ *and zero annihilates with tensor,* $U \otimes 0 = 0$*.*

A semiring where $(U, \otimes, \mathrm{I})$ *is commutative and* $(U, \oplus, 0)$ *idempotent, is called a commutative idempotent semiring. Idempotent semirings have a natural partial order defined as* $U \leq V$ *iff* $U \oplus V = V$*.*

The natural partial order over idempotent semirings, defined above, is used to define other algebraic properties. Kozen uses this preorder to provide the first sound and complete axiomatisation of a Kleene algebra [42]. The trick is to introduce rules that establish that the Kleene star is the least fixed point of a monotonic map. Furthermore, Kozen shows that the commutative idempotent semiring structure of Boolean test form a subalgebra of a Kleene algebra [43]. Kleene algebras with tests have proven to be powerful tools in conventional program verification. In this calculus commutative Kleene algebras with tests appear, where the Boolean test embedded are SPARQL filters.

Definition 7 (Kleene algebra with tests)**.** *A commutative Kleene algebra* $(U, *, \otimes, \oplus, \mathrm{I}, 0)$ *is a structure such that* $(U, \otimes, \oplus, \mathrm{I}, 0)$ *is a commutative idempotent semiring and terms*

of the form $V \otimes *U$ are the least fixed point of a (monotone) map $F \colon X \mapsto V \oplus (U \otimes X)$, which is characterised by the following property.

$$F(W) \leq W \qquad \textit{if and only if} \qquad V \otimes *U \leq W$$

A Kleene algebra with tests $(U, \phi, *, \otimes, \oplus, \mathrm{I}, 0, \neg)$ is such that $(U, *, \otimes, \oplus, \mathrm{I}, 0)$ is a Kleene algebra and $(\phi, \otimes, \oplus, \mathrm{I}, 0, \neg)$ is a Boolean algebra, where $\phi \subseteq U$.

The algebra here extends a Kleene algebra with tests. The extensions allow URIs and literals to be accessed in data, interactions to take place, and deals with interleaving concurrency.

Using the natural order, least upper bounds can be defined. It is immediate from the definition of an idempotent semiring that the choice of two processes is their least upper bound, and both existential quantification and iteration are least upper bounds.

Least upper bounds distribute over the tensor product, which is the characteristic property of a quantale [3, 34]. An instance of this quantale law is the distributivity of choice over tensor. As existential quantification and the Kleene star are least upper bounds, they distribute over tensor. All this is formulated below.

Definition 8 (Least upper bounds)**.** *Let $\bigvee \{U_i \mid i \in I\}$, the least upper bound of a set of processes indexed by I, be defined as follows.*

$$\bigvee \{U_i \mid i \in I\} \leq W \qquad \textit{iff} \qquad U_i \leq W, \textit{ for all } i \in I$$

The quantale law $\bigvee \{U_i \mid i \in I\} \otimes V = \bigvee \{U_i \otimes V \mid i \in I\}$ states that least upper bounds distribute over tensor.

Existential quantification is a least upper bound such that $\exists x.U = \bigvee \{U\{{}^{v}/_{x}\} \mid v \in \mathcal{V}\}$, where $\mathcal{V}$ is the set of values the quantification ranges over (URIs or literals).

The $*$-continuity property of the Kleene star is such that $*U = \bigvee \{U^n \mid n \in \omega\}$, where U^n is defined inductively asa $U^0 \triangleq \mathrm{I}$ and $U^{n+1} \triangleq U \otimes U^n$.

Note a Kleene algebra with the $*$-continuity property, is stronger than a Kleene algebra [42]. It follows from $*$-continuity that the Kleene star is a least fixed point, but the converse is not automatic.

A consequence of the quantale law applied to existential quantification as a least upper bound is the following useful proposition. This property was used in the examples in the introduction to change the scope of existential quantification.

Proposition 5. *If $x \notin \mathrm{fv}(V)$ then $\exists x.U \otimes V = \exists x.(U \otimes V)$.*

Proof. Consider the case of distributivity of exists over tensor. Assume that $x \notin \mathrm{fn}(V)$. Hence the following reasoning holds.

$$\begin{array}{lll}
\exists x.U \otimes V & = \bigvee \{U\{{}^{v}/_{x}\} \mid v \in \mathcal{V}\} \otimes V & \text{since exists is a least uper bound} \\
& = \bigvee \{U\{{}^{v}/_{x}\} \otimes V \mid v \in \mathcal{V}\} & \text{by the quantale law} \\
& = \bigvee \{(U \otimes V)\{{}^{v}/_{x}\} \mid v \in \mathcal{V}\} & \text{since } x \notin \mathrm{fv}(V) \\
& = \exists x.(U \otimes V) & \text{since exists is a least uper bound}
\end{array}$$

Thus existential quantification distributes over tensor. □

Interaction is captured by the relationship between par and tensor. Like tensor par forms a commutative monoid. However, least upper bounds only partially distribute over par. Par is related to the other constructs of the calculus by the adjunction of linear negation. Consequently labels form a model for multiplicative linear logic.

Definition 9. *A model of multiplicative linear logic* $\left(A, \otimes, \parr, \mathrm{I}, \bot, (.)^{\bot}, \leq\right)$ *is such that* $(A, \otimes, \mathrm{I})$ *is a commutative monoid, if* $A \leq A'$ *and* $B \leq B'$ *then* $A \otimes B \leq A' \otimes B'$, $A^{\bot\bot} = A$, *and the adjunction* $B \otimes A \leq C^{\bot}$ *iff* $B \leq (C \otimes A)^{\bot}$ *holds. Furthermore,* $\bot = \mathrm{I}^{\bot}$ *and* $A \parr B = (A^{\bot} \otimes B^{\bot})^{\bot}$.

Finally, some algebraic properties characterise delay and interleaving. Interleaving is a commutative monoid, with unit 0. Interleaving also satisfies some characteristic inequalities. These inequalities are included in the following summary of the algebraic properties of the calculus.

Definition 10 (Algebraic properties)**.** *The algebra for the calculus is defined as the least equivalence relation over processes terms U, Boolean tests* ϕ *and labels A, such that following properties hold.*

- $(U, \phi, *, \otimes, \oplus, \mathrm{I}, 0, \neg, \leq)$ *forms a Kleene algebra with tests.*
- *Existential quantification is the least upper bound of substitutions for a variable, as in Definition 8.*
- *Iteration is the least upper bound of powers of processes, as in Definition 8.*
- *Least upper bounds distribute over tensor.*
- $\left(A, \otimes, \parr, \mathrm{I}, \bot, (.)^{\bot}, \leq\right)$ *is a model of multiplicative linear logic.*
- $(U, \parr, 0)$ *is a commutative monoid.*
- $(U \parr V) \leq U \parallel V$, $(U \otimes \tau V) \leq U \parallel V$, $(\tau U \otimes V) \leq U \parallel V$ *and* $\tau(U \parallel V) \leq \tau U \otimes \tau V$.

Combining the properties for interleaving results in the following disequalities.

$$\tau(U \otimes \tau V) \leq \tau U \otimes \tau V \qquad \tau(\tau U \otimes V) \leq \tau U \otimes \tau V \qquad \tau(U \parr V) \leq \tau U \otimes \tau V$$

Notice that the algebraic structure of process interleaving is not made explicit in this definition. Interleaving could also be modelled indirectly by treating τ as a modality with the above properties and a retrodictive adjoint, similarly to Moortgat [52]. Such a system is a significant departure from this presentation of the calculus, so is left as further work.

7.2. Soundness of the algebra

We now prove that the algebra for the calculus is sound, by verifying that every algebraic property in Definition 10 holds with respect to bisimulation. Thus any application of the operators in the algebra preserve operational equivalence.

Theorem 6 (Soundness of the algebra)**.** *If* $U = V$ *in the algebra, then* $U \sim V$.

Proof. Fistly consider the cases which establish that $(P, \otimes, \oplus, \mathrm{I}, 0)$ is a commutative idempotent semiring.

Consider the case of idempotency of choice. Assume that $U \rhd P$, which holds iff $U \oplus U \rhd P$. Hence the least symmetric relation $\sim_0$ such that $U \oplus U \sim_0 U$ is a bisimulation.

Consider the case of associativity of choice. There are three cases to consider. Firstly, assume $U \rhd P$ holds. then the following trees are interchangeable.

$$\dfrac{\dfrac{U \rhd P}{U \oplus V \rhd P}}{(U \oplus V) \oplus W \rhd P} \quad \text{iff} \quad \dfrac{U \rhd P}{U \oplus (V \oplus W) \rhd P}$$

Secondly, if $V \rhd Q$ then the following trees are interchangeable.

$$\dfrac{\dfrac{V \rhd Q}{U \oplus V \rhd Q}}{(U \oplus V) \oplus W \rhd Q} \quad \text{iff} \quad \dfrac{\dfrac{V \rhd Q}{V \oplus W \rhd Q}}{U \oplus (V \oplus W) \rhd Q}$$

The third case is symmetric to the first case. Hence the least equivalence relation such that $U \oplus (V \oplus W) \sim_0 (U \oplus V) \oplus W$ is a bisimulation.

Consider the case of commutativity of choice. Assume that $U \rhd P$ holds. By choose left $U \oplus V \rhd P$ and by choose right $V \oplus U \rhd P$. Hence the least symmetric relation $\sim_0$ such that $U \oplus V \sim_0 V \oplus U$ is a bisimulation.

Consider the case of the unit of choice. Assume that $U \rhd P$ holds. 0 cannot make a labelled transition, hence $U \oplus 0 \rhd P$ by left choice. Hence the least equivalence relation $\sim_0$ such that $U \oplus 0 \sim_0 U$ is a bisimulation.

Consider the case of associativity of tensor. Assuming that $U \rhd P$, $V \rhd Q$ and $W \rhd R$, the following proof trees are interchangeable.

$$\dfrac{U \rhd P \quad \dfrac{V \rhd Q \quad W \rhd R}{V \otimes W \rhd Q \otimes R}}{U \otimes (V \otimes W) \rhd P \otimes (Q \otimes R)} \quad \text{iff} \quad \dfrac{\dfrac{U \rhd P \quad V \rhd Q}{U \otimes V \rhd P \otimes Q} \quad W \rhd R}{(U \otimes V) \otimes W \rhd (P \otimes Q) \otimes R}$$

By the right structural congruence, $(P \otimes Q) \otimes R \equiv P \otimes (Q \otimes R)$. Hence the least equivalence $\sim_0$ such that $U \otimes (V \otimes W) \sim_0 (U \otimes V) \otimes W$ is a bisimulation.

Consider the case of commutativity of tensor. Assume that $U \rhd P$ and $V \rhd Q$ hold. Now, by the tensor rule, $U \otimes V \rhd P \otimes Q$ and $V \otimes U \rhd Q \otimes P$, and $P \otimes Q \equiv Q \otimes P$, by the right structural congruence. Hence commutivity of tensor is a bisimulation.

Consider the unit of tensor. Assume that $U \rhd P$. By the tensor rule, $U \otimes \mathrm{I} \rhd P \otimes \mathrm{I}$ and $P \otimes \mathrm{I} \equiv P$, by the right structural congruence. Hence the least equivalence relation such that $P \otimes \mathrm{I}$ is a bisimulation.

Consider the case of distributivity. Without loss of generality, assume that $U \rhd P$ and $V \rhd Q$. The following proof trees are interchangeable.

$$\dfrac{U \rhd P \quad \dfrac{V \rhd Q}{V \oplus W \rhd Q}}{U \otimes (V \oplus W) \rhd P \otimes Q} \quad \text{iff} \quad \dfrac{\dfrac{U \rhd P \quad V \rhd Q}{U \otimes V \rhd P \otimes Q}}{(U \otimes V) \oplus (U \otimes W) \rhd P \otimes Q}$$

Therefore the least equivalence relation $\sim_0$, such that $U \otimes (V \oplus W) \sim_0 (U \otimes V) \oplus (U \otimes W)$ is a bisimulation.

Consider the case of annihilation. Suppose that $U \otimes 0$ makes a transition. Then $U \rhd P$ and $0 \rhd Q$, for some Q, but 0 makes no labelled transition so no such Q exist, yielding a contradiction. Hence, the least relation $\sim_0$ such that $U \otimes 0 \sim_0 0$ is a bisimulation.

Consider the quantale law. Suppose that $\{P_i \mid i \in I\}$ is a set of processes indexed by I. By definition of a least upper bound, $\bigvee \{P_i \mid i \in I\} \leq W$ iff for all $i \in I$, $P_i \leq W$.

Assume that $P_i \otimes Q \rhd Z$. then there exists X, Y such that $Z \equiv X \otimes Y$ and $P_i \rhd X$, $Q \rhd Y$. Thus, by definition of a least upper bound, there exists X' such that $X' \sim X$ and $\bigvee \{P_i \mid i \in I\} \rhd X'$ thus $\bigvee \{P_i \mid i \in I\} \otimes Q \rhd X' \otimes Y$ and $X' \otimes Y \sim Z$. Therefore $\bigvee \{P_i \otimes Q \mid i \in I\} \leq \bigvee \{P_i \mid i \in I\} \otimes Q$, i.e. $\bigvee \{P_i \mid i \in I\} \otimes Q$ is an upper bound for $\{P_i \otimes Q \mid i \in I\}$.

Now assume that for all $i \in I$, $P_i \otimes Q \leq W$. Hence for all $i \in I$, if $P_i \otimes Q \rhd X \otimes Y$ then there exists Z such that $Z \sim X \otimes Y$ and $W \rhd Z$. Now suppose that $\bigvee \{P_i \mid i \in I\} \otimes Q \rhd X \otimes Y$ such that $\bigvee \{P_i \mid i \in I\} \rhd X$ and $Q \rhd Y$. Since is the least upper bound the exists some $i \in I$ such that $P_i \rhd X$. Hence $P_i \otimes Q \rhd X \otimes Y$. Thus, by the assumption $W \rhd Z$ such that $Z \sim X \otimes Y$. Hence $\bigvee \{P_i \mid i \in I\} \otimes Q \leq \bigvee \{P_i \otimes Q \mid i \in I\}$. Therefore $\bigvee \{P_i \mid i \in I\} \otimes Q \sim \bigvee \{P_i \otimes Q \mid i \in I\}$ as required.

Now consider existential quantification as a least upper bound. If $U\{^v/_x\} \rhd X$ then $\exists x.U \rhd X$, hence $U\{^v/_x\} \leq \exists x.U$. Conversely, assume that for all v, $U\{^v/_x\} \leq W$. Now, $\exists x.U \rhd X$ only if for some v, $U\{^v/_x\} \rhd X$. Hence, by the assumption, $W \rhd Y$, where $X \sim Y$. Thus $\exists x.U \leq W$.

Consider the $*$-continuity property of the Kleene star. Clearly if $U^0 \rhd \mathrm{I}$ then $*U \rhd \mathrm{I}$, by weakening. Now assume that $U^n \leq *U$ so if $U^n \rhd Y$ then there exists Y' such that $*U \rhd Y'$ and $Y \sim Y'$. Assume that $U \rhd X$ hence the first commitment below yields the second commitment.

$$\frac{U \rhd X \quad U^n \rhd Y}{U^{n+1} \rhd X \otimes Y} \qquad \text{yields} \qquad \frac{\dfrac{U \rhd X}{*U \rhd X} \quad *U \rhd Y'}{\dfrac{*U \otimes *U \rhd X \otimes Y'}{*U \rhd X \otimes Y'}}$$

Furthermore, $X \otimes Y \sim X \otimes Y'$ hence $U^{n+1} \leq *U$. Thus by induction $U^n \leq *U$ for all n.

Now assume that for all n, $U^n \leq W$ and consider $*U$. If $*U \rhd \mathrm{I}$, by weakening, then $U^0 \rhd \mathrm{I}$. If $U \rhd X$, then $*U \rhd X$, by derelication, hence $U^1 \rhd X$. Now assume that if $*U \rhd X$ and $*U \rhd Y$, then $U^m \rhd X$ and $U^n \rhd Y$. Now assume that $*U \rhd X \otimes Y$ follows from contraction and tensor. Note that, by induction on n, $U^m \otimes U^n \sim U^{m+n}$. The base case follows from the unit of multiplication, since $U^m \otimes U^0 \sim U^m$. The induction step follows since $U^m \otimes U^{n+1} \sim U^{m+n} \otimes U$. Hence, by the inducation hypothesis, $U^{m+n} \rhd X \otimes Y$. Hence by induction $*U \leq W$

Consider the case of the labels. Let $\sim_0$ be the least equivalence which is a model of multiplicative linear logic over labels. By the same argument as for semirings before $(A, \otimes, \mathrm{I})$ is a commutative monoid. Since labels have no continuations the preorder simplifies to $A \leq B$ iff $B \rhd A$. Hence it is easy to check the conditions. If $A \otimes B \leq C^\perp$ iff $B \leq A^\perp \parr C^\perp$, where the right inequality follows from cut applied to $A^\perp \rhd A^\perp$ and $C^\perp \rhd A \otimes B$. Similarly, $A \rhd A$ iff $A^{\perp\perp} \rhd A$, by the De Morgan properties. Hence $\sim_0$ is a bisimulation.

Consider 0 as the unit of interleaving. Let $\sim_0$ be the least symmetric relation such that $P \parallel 0 \sim_0 P$. Assume that $P \rhd A \otimes \tau Q$. The only commitment $P \parallel 0$ can perform is

$P \parallel 0 \rhd A \otimes \tau\,(Q \parallel 0)$ and $Q \parallel 0 \sim_0 Q$. Hence $\sim_0$ is a bisimulation.

Consider the commutivity of interleaving. Let $\sim_0$ be the lest symmetric relation such that $P \parallel Q \sim_0 Q \parallel P$. There are four cases to check. Two for the left and right action rules and two for par. One case for actions is presented below.

$$\dfrac{\dfrac{\tau P \rhd \tau P \quad Q \rhd A \otimes \tau Q'}{\tau P \otimes Q \rhd A \otimes \tau\,(P \parallel Q')}}{P \parallel Q \rhd A \otimes \tau\,(Q' \parallel P)} \quad \text{iff} \quad \dfrac{\dfrac{\tau P \rhd \tau P \quad Q \rhd A \otimes \tau Q'}{\tau P \otimes Q \rhd A \otimes \tau\,(Q' \parallel P)}}{Q \parallel P \rhd A \otimes \tau\,(Q' \parallel P)}$$

Furthermore $P \parallel Q' \sim_0 Q \parallel P$. All other cases make similar use of the symmetry of rules. Thus $\sim_0$ is a bisimulation.

Consider the associativity of interleaving. Let $\sim_0$ be the least symmetric relation such that $P \parallel (Q \parallel R) \sim_0 (P \parallel Q) \parallel R$. There are many cases to check either all three synchronise, two can synchronise or one process can act independently. One case for an right independent action is shown.

$$\dfrac{\dfrac{\tau\,(P \parallel Q) \rhd \tau\,(P \parallel Q) \quad R \rhd A \otimes \tau R'}{\tau\,(P \parallel Q) \otimes R \rhd A \otimes \tau\,(P \parallel Q \parallel R')}}{(P \parallel Q) \parallel R \rhd A \otimes \tau\,(P \parallel Q \parallel R')} \quad \text{iff} \quad \dfrac{\tau P \rhd \tau P \quad \dfrac{\tau Q \rhd \tau Q \quad R \rhd A \otimes \tau R'}{\tau Q \otimes R \rhd A \otimes \tau\,(Q \parallel R')}}{\dfrac{P \parallel (Q \parallel R) \rhd A \otimes \tau\,(P \parallel Q \parallel R')}{P \parallel (Q \parallel R) \rhd A \otimes \tau\,(P \parallel Q \parallel R')}}$$

The remaining cases follow by similar restructuring of proof trees. Checking all cases established that $\sim_0$ is a bisimulation.

The full cases analysis for a preliminary version of this calculus appears in the thesis of the first author [37]. □

An immediate consequence of Theorem 6 and Theorem 3 is that the algebra is also sound with respect to contextual equivalence. Thus, the algebra respects the natural notion of operational equivalence.

Future work is planned to establish completeness of the algebra with respect to bisimulation. Completeness of the algebra would establish that if two processes are bisimilar then they can be proven to be equivalent using the algebra. Theorem 4 would then be invoked to prove that the algebra is complete with respect to contextual equivalence.

The ultimate aim is that the three semantics of the calculus, the reduction semantics, labelled transition semantics and algebraic semantics coincide. Note however, the completeness of the algebra may prove to be evasive due to the use of an interleaving semantics for parallel composition. Thus future work will also investigate the soundness and completeness of an algebra for related calculi with non-interleaving semantics [18].

8. Conclusion

This work employs operational equivalences to extend algebras for rewriting queries to a broader setting including concurrent updates. The high level langauge introduced is a generalisation of several recent W3C recommendations [30, 25]. The language combines powerful queries and updates for reading and writing Linked Data with

constructs for specifying concurrent systems, such as continuations and interleaving parallel composition.

The operational semantics of the calculus is expressive but concise. Both the reduction system and the labelled transition systems consists of 14 rules, and are presented using commitment relations. The choice of presentation means that familiar logical concepts, such as existential quantification, are defined in the expected fashion. The algebra confirms expected properties of constructs, and reveals that the system extends well known systems including multiplicative linear logic and Kleene algebras with tests. The algebra is proven to be sound.

The proofs are syntactic, hence there are many cases to check, but there are interesting features. Theorem 2 and Theorem 6 feature mixed induction and coinduction to handle the Kleene star. Mixed induction and coinduction is expected in a substantial system that mixes operational behaviour and data, and is handled by modern proof assistants. Some neat algebraic characterisations of constructs, such as the distributivity of least upper bounds over tensor simplify the proof of the algebra.

The choice of presentation of the reduction system and labelled transition system enables a succinct comparison of the systems. Trivial syntactic translations are avoided, so the soundness proof for bisimulation focusses on the essence of the proof. Soundness and completeness rely on the admissibility of a the cut rule, Theorem 1. The admissibility proof involves identity, commutative and principal cases, similarly to a cut elimination proof in proof theory. Soundness also relies of a context closure result that establishes that bisimulation is a congruence.

Finally, notice that the algebra introduced has an advantage over traditional relational algebra. It has been proven to be correct for environments with concurrently running and interacting processes. Thus the algebra can be applied in concurrent settings such as for a domain specific concurrent programming language for coordinating Linked Data applications in a cluster of machines.

Acknowledgements. We thank the anonymous reviewers for their constructive feedback.

References

[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[2] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.

[3] Samson Abramsky and Steven Vickers. Quantales, observational logic and process semantics. *Mathematical Structures in Computer Science*, 3(02):161–227, 1993.

[4] John Warner Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *Proceedings of the International Conference on Information Processing*, pages 125–132. Oldenbourg, Munich and Butterworth, London, 1959.

[5] Gianluigi Bellin and Philip J. Scott. On the π-calculus and Linear Logic. *Theoretical Computer Science*, 135:11–65, 1994.

[6] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[7] Tim Berners-Lee. Linked Data. *International Journal on Semantic Web and Information Systems*, 4(2):1, 2006.

[8] Tim Berners-Lee. Read-Write Linked Data. personal note, 2013.

[9] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3):249–269, 2008.

[10] Paul V. Biron and Ashok Malhotra. *XML Schema part 2: Datatypes Second Edition*. W3C, MIT, MA, 2004.

[11] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[12] Christian Bizer. The emerging Web of Linked Data. *IEEE Intelligent Systems*, 24:87–92, 2009.

[13] Christian Bizer et al. DBpedia: A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[14] Gérard Boudol and Ilaria Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In J. de Bakker, W. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 411–427. Springer, 1989.

[15] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication and mobility control in boxed ambients. *Information and Computation*, 202:39–86, 2005.

[16] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

[17] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):247–267, 2005.

[18] Gabriel Ciobanu and Ross Horne. A provenance tracking model for data updates. In *In Proceedings FOCLASA 11*, volume 91 of *EPTCS*, pages 31–44, 2012.

[19] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[21] Mariangiola Dezani, Ross Horne, and Vladimiro Sassone. Tracing where and who provenance in Linked Data: a calculus. *Theoretical Computer Science*, 2012.

[22] Roy T. Fielding and Richard N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

[23] Jean Gallier. Constructive logics. Part II: Linear Logic and Proof Nets. Research Report PR2-RR-9, Digital Equipment Corporation, Paris, 1991.

[24] Jean H. Gallier. Constructive logics part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[25] Paul Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 update. Recommendation REC-sparql11-update-20130321, W3C, MIT, MA, 2013.

[26] Gerhard Gentzen. Untersuchungen uber das logische schließen. *Mathematische Zeitschrift*, 39(2):176–210, 1934.

[27] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–112, 1987.

[28] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS '07*, pages 31–40. ACM, 2007.

[29] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Compututational Logic*, 8, 2007.

[30] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. Recommendation REC-sparql11-query-20130321, W3C, MIT, MA, 2013.

[31] Olaf Hartig et al. Executing SPARQL Queries over the Web of Linked Data. In A. Bernstain et al., editors, *The Semantic Web – ISWC 2009, Chantilly, VA*, volume 5823, pages 293–309. Springer, 2009.

[32] Michael Hausenblas. Exploiting Linked Data to build Web applications. *Internet Computing, IEEE*, 13(4):68–73, 2009.

[33] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[34] C. A. R. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene algebra. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009, Bologna, Italy*, volume 5710, pages 399–414. Springer, 2009.

[35] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of Intuitionistic Linear Logic. *Information and Computation*, 110(2):327–365, 1994.

[36] Ross Horne. *Programming Languages and Principles for Read–Write Linked Data*. PhD thesis, School of Electronics and Computer Science, University of Southampton, 2011.

[37] Ross Horne and Vladimiro Sassone. A verified algebra for Linked Data. In António Ravara and Mohammad Reza Mousavi, editors, *In proceedings of FOCLASA 10. Aachen, Germany. 10 August*, volume 58, pages 20–33. Electronic Proceedings in Theoretical Computer Science, 2011.

[38] Ross Horne, Vladimiro Sassone, and Nicholas Gibbins. Operational semantics for SPARQL Update. In Jeff Z. Pan, Huajun Chen, Hong-Gee Kim, Juanzi Li, Zhe Wu, Ian Horrocks, Riichiro Mizoguchi, and Zhaohui Wu, editors, *1st Joint International Semantic Technology Conference. Hangzhou, China. 4–7th December*, number 7185 in Lecture Notes in Computer Science, pages 242–257. Springer, 2011.

[39] Alan Jeffrey and Peter Patel-Schneider. Integrity constraints for Linked Data. In *Proceedings of 24th International Workshop on Description Logics, Barcelona, Spain, 13-16th July*, pages 521–531, 2011.

[40] Graham Klyne and Jeremy Carroll. *Resource Description Framework: Concepts and Abstract Syntax*. W3C, MIT, MA, 2004.

[41] Naoki Kobayashi and Akinori Yonezawa. ACL - a concurrent Linear Logic programming paradigm. In *Proceedings of the 1993 International Logic Programming Symposium*, pages 279–294. MIT Press, 1993.

[42] Dexter Kozen. On Kleene algebras and closed semirings. In Rovan, editor, *Proceedings on Mathematical Foundations of Computer Science*, volume 452, pages 26–47. Springer-Verlag, 1990.

[43] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programing Languages and Systems*, 19:427–443, 1997.

[44] Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres. On blank nodes. In *International Semantic Web Conference (1)*, pages 421–437, 2011.

[45] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[46] Massimo Merro and Matthew Hennessy. Bisimulation congruences in safe ambients. In *Principles of programming languages*, pages 71–80. ACM, 2002.

[47] Massimo Merro and Francesco Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52:961–1023, November 2005.

[48] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, April 1992.

[49] Robin Milner. The polyadic π-calculus: A tutorial. In F.L. Bauer, W. Brauer, and H. Schwichttenburg, editors, *Logic and Algebra in Specification*. Springer, New York, 1993.

[50] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–40, 1992.

[51] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Berlin / Heidelberg, 1992.

[52] Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, pages 349–385, 1996.

[53] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, 2009.

[54] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[55] Vaughan R. Pratt. Higher dimensional automata revisited. *Mathematical Structures in Computer Science*, 10(4):525–548, 2000.

[56] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *The Semantic Web: Research and Applications*, pages 524–538. Springer, 2008.

[57] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.

[58] Andy Seaborne and Geetha Manjunath. *SPARQL/Update: A language for updating RDF graphs*. Hewlett–Packard Laboritories, Bristol, 2007.

[59] Nigel Shadbolt, Kieron O'Hara, Tim Berners-Lee, Nicholas Gibbins, Hugh Glaser, Wendy Hall, and m. c. schraefel. Linked open government data: Lessons from data.gov.uk. *IEEE Intelligent Systems*, 27(3):16–24, 2012.

[60] Philip Stutz, Abraham Bernstein, and William Cohen. Signal/collect: Graph algorithms for the (semantic) web. In Peter Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web – ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 764–780. Springer Berlin / Heidelberg, 2010.

[61] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank Harmelen. Scalable distributed reasoning using MapReduce. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 634–649, Berlin, Heidelberg, 2009. Springer-Verlag.