A Verified Algebra for Read-Write Linked Data

Ross Horne, Vladimiro Sassone

Electronics and Computer Science, University of Southampton, United Kingdom

Abstract

The aim of this work is to verify an algebra for high level languages for reading and writing Linked Data. Linked Data refers to a collection of standards which aim to enhance the world's data, by interlinking datasets through the Web. The starting point is as simple as using URIs as global identifiers in data, but the technical challenges of managing data in this distributed setting are immense. An algebra is an essential contribution to this application domain.

To verify the algebra several useful things are established. A high level language is defined that concisely captures query and update languages for Linked Data. The language is provided with a concise operational semantics. The natural notions of equivalence, contextual equivalence, is shown to coincide with the bisimulation proof technique. Ultimately, bisimulation allows the algebra proven to be correct. Some novel techniques are used in establishing these results.

Keywords: operational semantics, bisimulation, Linked Data

1. Introduction

This work focusses on high level languages for reading and writing data on the Web [5]. This powerful emerging movement in computer science is commonly referred to as the Web of Data or Linked Data due to the emphasis on using URIs to establish links across distributed datasets [9]. The movement is gaining considerable momentum as major organisations including the UK and US governments, begin to adopt associated technologies for valuable data [33].

The key feature of Linked Data is the URI. The URI is just a standardised and globally recognised identifier. Instead of publishing formatted documents, as with traditional Web sites, data is published directly by organisations. Instead of URIs appearing as hyperlinks in a formatted document, the relationships between resources identified by URIs is conveyed by the data. Using URIs in this way is a small conceptual shift. However, the shift enables new opportunities by interlinking datasets from multiple organisations.

Email addresses: rjh06r@ecs.soton.ac.uk (Ross Horne), vs@ecs.soton.ac.uk (Vladimiro Sassone)

The traditional setting for data is well understood. Without a global naming system such as the URI, each dataset would need to use its own naming system. Typically, in that case each dataset is disjoint. This is referred to as a closed system, since the boundaries of the data are known, and classical negation can be used to determine whether some data does not appear in a dataset. Also, schemata can be employed ensure consistency of the data.

The Linked Data setting presents significant new challenges for engineers. In contrast to traditional databases, the presence of URIs as a global naming system, connects distributed data sets. Protocols can be used to obtain data from multiple sources based on URIs, which appear in data [23]. For instance, a request may be sent to a URI to directly obtain some data about that URI. In this setting, there is no guarantee that the protocol finds all relevant data. There may always be relevant data on the Web which was not obtained during this process. Thus, in general, optimal query results cannot be obtained nor classical negation applied. Furthermore, due to decentralisation, schemata which constrain data cannot be enforced globally.

Several technologies for Linked Data standardised by the World Wide Web Consortium have found widespread use. These technologies include a light data format called the Resource Description Format (RDF [24]), and the SPARQL protocol and RDF query language [34]. RDF is a more loosely structured data format than XML. RDF also resembles the universal tool of simple sentences in natural language, consisting of a subject, predicate, and an object, where the predicate is used like a verb in natural language. This allows diverse data from different sources to be lifted to RDF. The idea is that data from many datasets can then be queried using one mechanism. Some smart techniques have been employed to power such queries [38, 37].

The authors provided the first operational semantics for RDF and SPARQL Query [21]. This presentation goes further by also covering SPARQL Update, which is a W3C candidate recommendation [15]. SPARQL Update allows updates to be specified which delete data, insert data and constrain updates according to some query. The first operational semantics for SPARQL Update was also provided by the authors [22]. This work presents a high level language which internalises RDF, SPARQL Query and SPARQL Update with some constructs for concurrency. The difference between the update language and the query language is only one axiom, so it makes sense to consider this more general setting. In this sense a new high level language is proposed.

In this work a natural notion of operational equivalence is used to assign semantics to the terms of a Linked Data process calculus. Furthermore, the calculus is axiomatised using algebraic structures like semirings, and such an algebra agrees with expected operational equivalences. The equations derived from algebra can be used to rewrite processes via equational reasoning, which may enable quick implementations, essential optimisations and novel programming techniques. Optimisations may for instance include finding normal forms for the distribution of updates across collections of datasets, a key problem for Linked Data [17].

To express the operational semantics of languages for Linked Data, a new framework for operational semantics has been introduced. In many ways, the framework goes beyond traditional frameworks for concurrency, such as the π-calculus [31], due to the powerful atomic actions which must be expressed. Indeed the framework raises significant questions about operations for concurrency which are often taken for granted,

such as interleaving parallel composition 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. Readers who do not require motivation 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. Finally, section 7 introduces the algebra over the language and proves its correctness.

2. A Bridge Between Operational Semantics and Linked Data

This paper connects 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 be interested in the implications for Linked Data, while others are interested in the novel framework involved. We aim at addressing both audiences, 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 [2]. Axioms and rules of the calculus are then expressed in the style of natural deduction. In a rule several premises, above the line, lead to a conclusion, below the line.

The deductive system presented specifies an operational semantics for a process. Traditionally operational semantics are expressed in two ways. The first approach is to define a reduction system; the second is to define a labelled transition system. A reduction system directly defines complete transitions for a process. The process on the left of a transition 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, represented by a label which constrains the context in which a transition takes place. Traditionally, a labelled transition system it verified by proving that the natural notion of equivalence in the labelled transition system is is sound and complete with respect to the notion of equivalence in the reduction system [35, 29]. For a similar calculus soundness has been proven in this traditional fashion [20].

The novelty of the approach presented here is that both the reduction system and the labelled transition system are express using almost identical deductive systems. The key difference is that the labelled transition system is forced to use a cut rule to achieve the effect of the interaction of labels. Cut rules are transitivity properties of logical systems. The study of their cut elimination, the process of rewriting deductions to a form which does not use cut, fundamentally characterises the dynamics of a deductive system [14]. The key feature of the reduction system is that it does not use the cut rule. This suggests striking connections between process calculi and logic, through the proof of the soundness and completeness of the labelled transition system with respect to the reduction system.

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 and SPARQL as modelled in this calculus. The queries and updates are executed according to the 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 Linked Data. This particular dataset is taken from DBpedia, which lifts Wikipedia to RDF [10].[1] The data concerns a retired footballer called Joe Armstrong. Armstrong is assigned a URI *Armstrong* to identify him in Linked Data. The RDF data below is presented as triples. A triple consists of a subject, predicate and object. The subjects are URIs, *Armstrong* in this case. The predicates indicates how the subject is related to the object. Predicates are also URIs which are chosen from 'metadata' vocabularies, such as *foaf:name* which is drawn from the FOAF vocabulary. The object can be either a URI or a literal data value. The object of a triple with the *dbp:birthDate* predicate is a date literal; while the football club Gateshead F.C. is identified by a URI.

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

Note a predicate *rdf:type* from the core RDF vocabulary is used. This predicate 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, which 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. The term

[1] Clicking the URIs and predicates below will direct the reader's browser to their DBpedia definitions.

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

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

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 select one triple and binds variables to URIs through the existential (or select) construct, so as to perform the correct update. After the update, the process commits to the state below.

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

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 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. They 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 and the $(\_)^{\perp}$ construct); the third triple inserts the updated data. Observe that the final component of the update term is a Boolean constraint on a's date of birth.

All of these components – the query, delete, insert and constraint – are performed in the same atomic step. This ensures that the insert takes place only if the query and constraint 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 constraints. According to this, the update in the previous section can be rewritten in the following form.

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

This simple rewrite make use of several rules, which we shall introduce formally and prove correct 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.

This new form corresponds to the first syntax for SPARQL Update proposed by HP Labs [36]. Here below is our example update in the HP Labs SPARQL Update syntax, where ? denotes the variable binding construct in a query.

```
DELETE { ?a  dbp:position res:Inside_forward }
INSERT { ?a  dbp:position res:Attacking_midfielder }
WHERE {
  ?a dbp:birthDate ?x
  FILTER (?x <= '01-01-1950')
}
```

This rewrite demonstrates a quick implementation of the update language. Updates in the calculus can be normalised, then rewritten to updates in the original language of HP Labs. This allows the calculus to be used as a high level language for updating RDF. A more powerful use of the calculus would be inside a compiler for SPARQL. The algebra can be used to rewrite updates to a normal form, while the operational semantics can be used to plan the execution of processes.

3.4. A Larger Example Using the Algebra

We 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* predicate or by combining *foaf:givenName* and *foaf:familyName* predicates.

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

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 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 application at hand. The query used in the transition above can be rewritten as the disjunction of four queries in a normal form.

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

The above rewrite uses many of the algebraic properties of the calculus. It uses the facts that select 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 [24, 34]. Here an alternative, more concise 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 [22]. The syntax borrows from both traditional process calculi, such as the π-calculus, and also fragments of linear logic [16]. This work presents a new approach and application for combining process calculi and linear logic [25, 19, 3].

4.1. An Abstract Syntax for RDF Triples

A commonly used syntax for RDF is called N3 [6]. It is simple and intuitive. Its main feature is that triples are expressed as URIs in subject, predicate and object form terminated by a full stop. The full stop is unnecessary in abstract syntax. Thus a triple is expressed as follows.

Armstrong dbp:position res:Inside_forward

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

$$
\begin{array}{rcll}
\phi & ::= & \mathrm{I} & \text{true} \\
& | & 0 & \text{false} \\
& | & \phi \oplus \phi & \text{or} \\
& | & \phi \otimes \phi & \text{and} \\
& | & \neg\phi & \text{not} \\
& | & \ldots & \text{etc.}
\end{array}
$$

a URI v literal or URI

x variable for URI or literal

$$
\begin{array}{rcll}
U & ::= & A & \text{label} \\
& | & \phi & \text{filter} \\
& | & U \oplus U & \text{choice} \\
& | & U \otimes U & \text{tensor} \\
& | & U \parr U & \text{par} \\
& | & \exists x.U & \text{exists} \\
& | & *U & \text{iteration} \\
& | & \tau U & \text{delay} \\
& | & U \| U & \text{interleave}
\end{array}
$$

Figure 1: The syntax of constraints (ϕ), labels (A) and processes (U).

Triples are built from atomic identifiers such as *a* and *Armstrong*. These represent either URIs or variables for URIs. The third identifier, the object, can either be a URI or a data value, called a literal. Literals include strings and dates as defined in the XML Schema Datatype specification [7].

Variables can appear in place of URIs, since triples can be used as patterns in processes where the variables are bound by some quantifier. Variables for literals, such as x, are distinguished from variables for URIs, such as a, which avoids some basic type errors [21].

This presentation makes several simplifications, in the interest of clarity. The main one is that '*blank nodes*,' which act as local identifiers in RDF, are not covered. Indeed, we assimilate blank nodes to fresh name quantification in the π-calculus [21]. Notice also that our use of triples as the basic datatype 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 [12].

4.2. A syntax for Boolean filters

Filters are well understood Boolean formulae over well understood terms. Typical constraints such as regular expressions over strings and equality tests are handled by the SPARQL Query Recommendation [34]. In SPARQL Query, constraints are used to filter query results. Constraints can be embedded in a process to control its behaviour.

Constraints are areas of a process where classical negation can be applied. In this area $\oplus$ acts as classical disjunction, $\otimes$ as classical conjunction, I is true and 0 is false. This style of embedding of Boolean constraints is similar to the use of tests in Kleene algebras [27].

4.3. The multiplicative connectives

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 two processes to occur synchronously in the same atomic step, but without communication. No information in either process can be used to interact with the other process. Tensor 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 unit of tensor coincides with I in the Boolean algebra.

Par and linear negation ($\parr$ and $^\perp$). Par is similar to tensor, since the two composed processes occur synchronously. However par forces communication, rather than forbidding it. The linear negation, or complement, of a process is the process which interacts "perfectly" with the original process. For instance a delete interacts perfectly with a stored triple. Thus stored triples are expressed as complemented triples. We do not regard par as a programming primitive, but just 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 the whole calculus, but complementation does not. The processes to which complementation is restricted form the labels, 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 existential quantifier ($\exists$). Existential quantifiers perform the role of the select keywords in SPARQL. Existential quantifiers bind 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 quantifier this calculus functions at a higher level of abstraction than SPARQL. At a lower level, a select query would indicate the values which are returned in some results format. The results format 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 time 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 is processes, since it is 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 of queries is indicated by the Kleene star, which allows zero or more copies of a query to be answered in one atomic step. Note that iteration differs from replication in common process calculi. Here all copies of an iterated query must be answered simultaneously using disjoint resources.

Explicit iteration allows some powerful updates to be expressed, which would normally require several steps to be expressed. For instance, the following update mixes iterated and non-iterated parts. It deletes the date of birth of people older than *Armstrong*.

$$\exists x.(|(\mathit{Armstrong}\ \mathit{dbp{:}birthDate}\ x)| * \exists a.\exists y.((a\ \mathit{dbp{:}birthDate}\ y)\ (y < x)))$$

Interleaving parallel composition. Interleaving parallel composition with communication is different from par and tensor. This kind of interleaving appears in the π-calculus for instance. Either the processes interact or one of the processes does nothing.

4.5. Abbreviations for common examples

A common operation is a query. A query is expressed as the process which deletes a triple then stores it in the next step. The triple must exists for the delete to occur, while the insert ensures that the triple persists. Thus a query $|A|$ is an abbreviation for the process $A \otimes \tau A^{\perp}$. Note that this implies a linear semantics for triples, in the precise sense that each can only be queried once in a time interval. Such abbreviation may be naïve in certain contexts and under some applications but is useful in the present paper, as 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.

5. An Operational Semantics for Read-Write Linked Data

This section defines an operational semantics for the calculus. Traditionally an operational semantics is expressed as a reduction system and a labelled transition system. In previous work a reduction system and a labelled transition system are presented [21]. The two systems were found to be intimately connected, thus the presentation here uses a single 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 commitments. A process can commit to a state which is more deterministic. This state can be so deterministic that at most one reduction or labelled transition is possible. Commitments provide a powerful framework for operational semantics.

Reductions are often expressed by defining a relation between states $P \longrightarrow Q$. The process on the left is the state before the reduction. The process on the right is the state after one step has occurred. Instead a special prefix operator τQ is be used, which indicates that in the next state Q occurs. Hence if $\triangleright$ is the relation which commits a

state to a more deterministic state, then the commitment $P \triangleright \tau Q$ commits P to the above reduction.

A similar principle applies to the labelled transitions. A labelled transition might typically be written as $P \xrightarrow{A} Q$, where P is the initial state which emits the label A to reach state Q. This is the same as stating that P can commit to the process which must do (or consume) A to reach the next state Q. A labelled transition is therefore expressed by the commitment $P \triangleright A \otimes \tau Q$.

Commitments were first proposed for the labelled transitions of the the polyadic π-calculus [30]. The commitment relation greatly simplified the presentation of the operational semantics. Furthermore, by expressing both the reductions and labelled transition in this way intimate connections are revealed. Most rules are the same. The main difference is how interactions occur.

Labelled transitions accumulate information expected from the context on the labels. Two processes can interact with each other if they offer complementary information on the label. Complementary information in linear logic is expressed as linear negation, thus the complement of a label A is $A^{\perp}$. The following special case of cut, obtained from the rules of Fig. 5, is reminiscent of interaction in CCS between a label and its co-label, as shown on the right for comparison.

$$\frac{\dfrac{P \triangleright A \otimes \tau P' \quad Q \triangleright A^{\perp} \otimes \tau Q'}{P \parr Q \triangleright \tau P' \otimes \tau Q'}}{P \parallel Q \triangleright \tau (P' \parallel Q')} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} (P' \parallel Q')}$$

Notice that we use both $\parr$ and $\parallel$ for concurrent composition. The former composition is synchronisation with interaction, while the latter also allows interleaving. A similar technique was used by Bergstra in ACP, where parallel composition is decomposed into a left action, a right action, and synchronisation with communication [4]. In our work, the latter role is taken by par, which behaves like the homonymous operator from linear logic.

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 for interaction, and this is expressed using par. Our reduction system deals with interactions similarly to proofs in cut free linear logic.

5.2. The left and right structural rules

The operational semantics relies on two structural congruences, one for each side of the rule. The effect of such congruences in our calculus is similar to the use of sequents in Gallier's version of linear logic [14]. It is also reminiscent of the approach of Kobayashi and Yonezawa [25], 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. It has the same effect as the exchange structural rules of linear logic, which allows formulae to move next to each perform interactions.

The structural congruence on the right of a commitment is given in Fig. 3. On the right, the exchange rules applies to tensor rather than par. Here there also are also for handling continuations. In particular, the tensor product of two delayed process is equivalent to their parallel composition delayed, an equivalence first derived in [21].

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

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.

(reflexivity)
$$P \rhd P$$

(tensor)
$$\frac{P \parr U \rhd P' \quad Q \parr V \rhd Q'}{P \parr Q \parr (U \otimes V) \rhd P' \otimes Q'}$$

(par)
$$\frac{P \rhd U \otimes P' \quad Q \rhd V \otimes Q'}{P \parr Q \rhd (U \parr V) \otimes P' \otimes Q'}$$

(complement)
$$\frac{P \rhd Q \otimes A}{P \parr A^{\bot} \rhd Q}$$

(choose left)
$$\frac{P \parr U \rhd Q}{P \parr (U \oplus V) \rhd Q}$$

(choose right)
$$\frac{P \parr V \rhd Q}{P \parr (U \oplus V) \rhd Q}$$

(exists)
$$\frac{P \parr U\{^{v}/_{x}\} \rhd Q}{P \parr \exists x.U \rhd Q}$$

(filter)
$$\frac{\vDash \phi}{\phi \rhd \mathrm{I}}$$

(weakening)
$$*U \rhd \mathrm{I}$$

(dereliction)
$$\frac{P \parr U \rhd Q}{P \parr *U \rhd Q}$$

(contraction)
$$\frac{P \parr (*U \otimes *U) \rhd Q}{P \parr *U \rhd Q}$$

(interact)
$$\frac{P \parr Q \parr R \rhd S}{P \parr (Q \parallel R) \rhd S}$$

(left action)
$$\frac{P \parr (Q \otimes \tau R) \rhd S}{P \parr (Q \parallel R) \rhd S}$$

(right action)
$$\frac{P \parr (\tau Q \otimes R) \rhd S}{P \parr (Q \parallel R) \rhd S}$$

Figure 4: Commitment rules for reductions

Observe that the standard De Morgan properties of linear logic are also handled by this structural congruence. Including these equivalences as primitive simplifies our definitions. Notice however that although $\oplus$ is commutative too, we prefer not to add that as a structural rule: we leave that equivalence to be established 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 axioms of the system are pure logical axioms, which state that the commitment relation is reflexive. Combines with the standard adjunction of linear negation interactions are captured. The interaction of stored triples with deletes

is an instance of reflexivity followed by the standard adjunction of linear negation. To see this consider the following reasoning.

$$\frac{(\mathit{Armstrong\ p{:}clubs\ Gateshead}) \triangleright \mathrm{I} \otimes (\mathit{Armstrong\ p{:}clubs\ Gateshead})}{(\mathit{Armstrong\ p{:}clubs\ Gateshead}) \parr (\mathit{Armstrong\ p{:}clubs\ Gateshead})^{\perp} \triangleright \mathrm{I}}$$

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

The tensor rule. The tensor rule is identical to the tensor rule in linear logic. The two parts of the tensor product are answered using separate results. 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, so features in most examples. In examples the symbol for the tensor product is often omitted.

The select rules. The select rule is essential for accessing URIs and literals in queries. The rule is exactly the rule for first order existential quantification in linear logic. 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 the subject of the query and the continuation process.

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

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

Note that the input of values in the π-calculus can be captured in this logical fashion. Indeed, with fresh name quantification (which models blank nodes in RDF), the whole of the π-calculus can be embedded in this framework [20].

The choose rule. The choose rule selects one of two branches. Since RDF is a very loose format, this is feature is essential to offer more than one way for a query or update to interact with data. The following example deletes either someone who knows Armstrong or someone who Armstrong knows.

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

In the above example the left branch was chosen.

Filters. Constraints are evaluated as axioms, where the built in functions and satisfaction relation $\vDash$ for the evaluation of constraints are left to the SPARQL recommendation [34]. Of course, we assume that $\vDash$ define a Boolean algebra of constraints. This example embeds a constraint in an update. The constraint ensures that the difference between two dates is less than 5 years.

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

The rules are applied resulting in x and y being instantiated with dates. The constraint is then evaluated separately by the filter axiom. Thus the above process can commit to the state below.

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

Synchronous iteration. Notice that all previous examples show updates which are applied only once to data. However, some updates are meant to be applied several times to data, even unboundedly many times. Such fact is explicitly indicated 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.

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

Note that the Kleene star based on a commutative operator, such as tensor, was first investigated by Conway [13].

Interleaving with interaction. Interleaving with interaction is the parallel composition normally found in process calculi with interaction. The left and right actions rules allow one of the processes to be delayed, while performs an action. The third rule synchronises the two processes so they interact. Thus interleaving either behaves like par or skips one process. For instance a process may have stored triples which are not used in the current interaction.

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

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

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

Figure 5: Commitment rules for labelled transitions

All the above examples hold using interleaving instead of par. However, the following example explains the subtlety of interleaving, due to tensor, which demands that par and interleaving are distinguished.

$$A \parallel (|\,A\,| \otimes B) \parallel A^{\perp} \parallel B^{\perp} \rhd \tau(A \parallel A^{\perp}) \qquad\qquad A \parallel (|A| \otimes B) \parallel A^{\perp} \parallel B^{\perp} \not\rhd \tau\,(|A|)$$

The first process exhibits a valid commitment. The single delete A is then delayed, and the two components of the tensor interact with the stored data. The second process is not a valid commitment, because $|A|$ and B do not proceed synchronously. Yet, it would be derivable if $\parr$ and $\parallel$ were collapsed to a single operation. Yes, the system would be simpler, but it would be wrong!

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. Instead the necessary information for the interaction is expressed as a label on the right of the commitment.

The rules for the labelled transition system are presented in Fig. 5. Most of them are special cases of those of the reduction system obtained with $\perp$ as the context. Thus we do not repeat the explanation of the rules. Instead we explain how labels are derived, and the effect of the cut rule on them.

5.5. The interaction of a stored triple and a delete

The axioms are used to initiate labels. The following stored triple interacts in a context which uses that stored triple. Here there is only one possible label.

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

For the following process the label can be instantiated in several ways. The labels which can appear correspond to the contexts the process can interact with.

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

The two processes above emit complementary labels. Hence they can interact using the cut rule. This results in the same commitment which was derived using the rules of the reduction system.

5.6. The interaction of multiple triples

Expressive queries and updates involve more than one triple in an interaction. The following process involves two queries and a continuation. Both the queries contribute to the label, while all parts contribute to the continuation process. (Remember here that $|A| \equiv A \otimes \tau A^{\perp}$.) This results in the following 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 follow is a labelled transition for two stored triples in parallel. Firstly the interleaving is converted to par. Then axioms are combined using the par rule.

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

De Morgan properties align the two labels. Therefore the two processes can interact using cut. The cut rule maintains the continuations of the first process. 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 achieve the same effect, cut can be applied twice, where each of the cuts matches one triple with part of the update. Both approaches are equivalent, and this will be used in the proofs of our main 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.

In this operational semantics the reduction system and the labelled transition system are intimately connected. The intermediate results highlight this connection, through a cut elimination theorem and a congruence result.

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 and 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 $P \triangleright \tau P'$ then there exists some Q' such that $Q \triangleright \tau Q'$ and $P'\ \mathcal{R}\ Q'$.*

Reduction closure by itself is not sufficient to generate a meaningful equivalence. 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 ensuing 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 $P \triangleright A \otimes \tau P'$ then there exists some Q' such that $Q \triangleright A \otimes \tau Q'$ 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.*

The definition of bisimulation must be verified to be correct with respect to contextual equivalence for this calculus, as done for other calculi [35, 28, 11]. Correctness of bisimulation is proven by the results in this section.

6.2. A Cut Elimination Result for Labels Only

The main difference between the reduction system and the labelled transition system is that the reduction system does not employ the cut rule. The two systems should therefore be compared by proving that a reduction which was derived using the cut rule can be derived without using the cut rule.

The cut rule is restricted to allowing only labels to be cut. Despite this restriction, the elimination 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 key case for par and tensor are harder and constitute the points of main interest in the proof.

Lemma 1 (Cut-elimination)**.** *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.

$$\frac{P \rhd A^{\perp} \otimes Q \quad A \rhd A}{P \parr A \rhd Q} \qquad \text{iff} \qquad \frac{P \rhd A^{\perp} \otimes Q}{P \parr A \rhd Q}$$

Thus identity cuts can be eliminated. This is the basis of an inductive proof of cut elimination, which proceeds by showing that in each case the application of the cut rule can be pushed o level up towards the leaves of the proof tree.

Commutative cases push the cut past a rule which does not take part in the interaction. All commutative cases are easy and have the same form. We show only one commutative case for interleaving here.

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

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

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.

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

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

$$\frac{P \rhd A \otimes B \otimes P' \quad \dfrac{U \rhd A^{\perp} \otimes Q \quad V \rhd B^{\perp} \otimes R}{U \parr V \rhd (A^{\perp} \parr B^{\perp}) \otimes Q \otimes R}}{P \parr U \parr V \rhd P' \otimes Q \otimes R}$$

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

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

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

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

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

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

All cases are covered hence, by induction, the cut rule can be eliminated. □

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.

Lemma 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.

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

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. We only verify the case for cut over par 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.

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

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 $P \parr R \sim_1 Q \parr R$. 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 $P_0 \parallel P_1 \sim_1 Q_0 \parallel Q_1$.

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

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 established by proving that bisimulation is a context equivalence. Soundness uses all lemmas in this section.

Theorem 1 (Soundness of bisimulation)**.** *If $P \sim Q$ then $P \simeq Q$.*

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

As expected for coinductive techniques, 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 2 (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 Lemma 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 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 [8], which shares similar aims to the approach explored here.

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

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

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 [26]. The trick is to introducing rules, which establish that the Kleene star is the least fixed point of a monotonic map. Furthermore, Kozen show that the commutative idempotent semiring structure of Boolean constraints form a subalgebra of a Kleene algebra [27]. 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 constraints embedded as filters in SPARQL are the tests.

Definition 7 (Kleene algebra with tests)**.** *A commutative Kleene algebra* $(P, *, \otimes, \oplus, \mathrm{I}, 0)$ *is a structure such that* $(P, \otimes, \oplus, \mathrm{I}, 0)$ *is a commutative idempotent semiring and* $V \otimes *U$ *is the least fixed point of the 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 $(P, \phi, *, \otimes, \oplus, \mathrm{I}, 0, \neg)$ *is such that* $(P, *, \otimes, \oplus, \mathrm{I}, 0)$ *is a Kleene algebra and* $(\phi, \otimes, \oplus, \mathrm{I}, 0, \neg)$ *is a Boolean algebra.*

The algebra here extends 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 choice is the least upper bound of two processes, 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 [1, 18]. 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 [26]. 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 3. *If* $x \notin \mathrm{fv}\,(Q)$ *then* $\exists x.P \otimes Q = \exists x.(P \otimes Q)$.

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

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

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 algebraic properties of the calculus.

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

- $(P, \phi, *, \otimes, \oplus, \mathrm{I}, 0, \neg, \leq)$ *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.*
- $(P, \parr, 0)$ *is a commutative monoid.*

- $(P \parr Q) \leq P \parallel Q$, $(P \otimes \tau Q) \leq P \parallel Q$, $(\tau P \otimes Q) \leq P \parallel Q$ *and* $\tau\,(P \parallel Q) \leq \tau P \otimes \tau Q$.

Combining the properties for interleaving results in the following disequalities.

$$\tau\,(P \otimes \tau Q) \leq \tau P \otimes \tau Q \qquad \tau\,(\tau P \otimes Q) \leq \tau P \otimes \tau Q \qquad \tau\,(P \parr Q) \leq \tau P \otimes \tau Q$$

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 [32]. 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 is 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 4 (Soundness of the algebra)**.** *If $P = Q$ in the algebra, then $P \sim Q$.*

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.

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

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

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

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.

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

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.

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

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.

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

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 [21]. □

An immediate consequence of Theorem 4 and Theorem 1 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 2 is then invoked to prove that the algebra is complete with respect to contextual equivalence. The aim is that the three semantics of the calculus, the reduction semantics, labelled transition semantics and algebraic semantics coincide.

8. Conclusion

This work employs operational equivalences to extend algebras for rewriting queries to a broader setting including updates. Our system is a generalisation of several real languages proposed by the W3C for interacting with Linked Data. The languages allow powerful queries and updates to be applied to Linked Data. The generalised calculus considers queries and updates over Linked Data in the context of high level programming constructs, including continuations and interleaving parallel composition.

The operational semantics of the calculus is concise. Both the reduction 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 algebraic properties confirm that expected properties hold, and reveal 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. Lemma 2 and Theorem 4 feature mixed induction and coinduction to handle the Kleene star. Mixed induction and coinduction is expected in a substantial system which mixes operational behaviour and data, and is handled by modern proof assistants such as Agda and Coq. Some neat algebraic characterisations of constructs, such as the distributivity of least upper bounds over tensor also make the proof of the algebra reasonable.

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 follows from a light cut elimination result, where cut can only be applied to labels, and a context lemma. As expected for cut elimination, the proof involves identity, commutative and key cases.

Finally, notice that the algebra generated has a big advantage over traditional relational algebra. It has been proven to be correct in an interactive environment with concurrently running processes. Thus the algebra can be applied in concurrent settings such as a server or a powerful programming language.

References

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

[2] 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.

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

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

[5] Tim Berners-Lee. Read-Write Linked Data. Personal view only, December 2010.

[6] 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.

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

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

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

[10] 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.

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

[12] 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.

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

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

[15] Paul Gearon, Alexandre Passant, and Axel Polleres. *SPARQL 1.1 Update*, May 2011.

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

[17] 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.

[18] 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.

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

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

[21] Ross Horne and Vladimiro Sassone. A verified algebra for Linked Data. In António Ravara and MohammadReza Mousavi, editors, *10th Workshop on the Foundations of Coordination Languages and Software Architectures. Aachen, Germany. 10th August*, volume 58, pages 20–33. Electronic Proceedings in Theoretical Computer Science, 2011.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

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

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

[30] 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.

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

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

[33] Tope Omitola et al. Put in your postcode, out comes the data: A case study. In Lora Aroyo et al., editors, *The Semantic Web: Research and Applications. 7th Extended Semantic Web Conference*, volume 6088, pages 318–332, Heraklion, Crete, May 30 – June 3 2010. Springer, Berlin/Heidelberg.

[34] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C, MIT, Cambridge, MA, 2008.

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

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

[37] 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.

[38] 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.