Integrating Positional and Slotted Knowledge on the Semantic Web⋆

Harold Boley

Institute for Information Technology – e-Business,
National Research Council of Canada,
Fredericton, NB, E3B 9W4, Canada
Harold.Boley AT nrc.gc.ca

Abstract. POSL is proposed as a Semantic Web language for e-Business knowledge interchange, reconciling Horn logic's positional and F-logic's slotted formulas for representing facts and rules on the Web, optionally referring to RDFS or OWL classes for order-sorted typing. The POSL semantics directly enhances Herbrand models for n-ary relations by accomodating slotted clause instantiation and ground equality, further restricted through signatures and types. Webizing uses URIs in the IETF form of N3 for individuals, relations, slots, and types. Webized atoms further permit the representation of F-logic objects and RDF descriptions as anchored slotted facts enhanced by rules. All POSL notions are exemplified using an e-Business use case in logistics. The online translator from POSL to OO RuleML and POSL engine OO jDREW have enabled three recent e-Business applications, in music filtering, information integration, and expert finding.

1 Introduction

This paper treats the design, syntax, semantics, implementation, and e-Business use of integrated positional and slotted knowledge on the Semantic Web. It

introduces the positional-slotted (POSL) language, which integrates Prolog's positional and Frame-logic's (F-logic's [KL89,YK03]) slotted knowledge representation for the Semantic Web via a uniform notion of 'relation' and 'object', permitting URIs in the style of Notation 3 (N3) for naming all language elements. POSL was also directly inspired by RDF [Hay04] and RuleML [Bol03]. It is proposed as a very-high-level interchange format for relational/object-oriented (e-Business) facts and rules, as exemplified here for a logistics use case. Facts correspond to relational tuples and rules generalize SQL views, together constituting the database foundation of e-Business. F-logic extensions of such facts and rules proceed from relational to object-oriented databases, hence to UML and MOF (e-Business) software specification. For F-logic objects and all other language constructs, URIs permit "webizing" [http://www.w3.org/DesignIssues/Webize.html], which is central to the Semantic Web and its e-Business use. Semantic Web rules thus in the short term will leverage the success of Business Rules [http://www.businessrulesgroup.org/brmanifesto.htm] and in the long term will leverage the success of Web Services for e-Business. Such motivation was also decisive for W3C's Working Group developing the Rule Interchange Format (RIF) [GHPM06,BK07], for which RuleML and POSL are major inputs [http://www.w3.org/2005/rules/wg.html].

Recent experience with the development of Semantic Web languages such as OWL [DS04] and SWSL [BBB+05] has shown the many advantages of studying expressive classes and formal semantics using a 'human-oriented' syntax layer above the XML level. Also, as pioneered by N3 [http://www.w3.org/2000/10/swap/Primer], a concise non-XML ASCII syntax is very useful in developing knowledge bases, which can then be parsed into some much more tedious XML markup such as RDF/XML for (distribution and) processing through the multitude of XML-aware tools. The same principles underlie the RIF Condition Language [http://http://www.w3.org/2005/rules/wg/wiki/Extensible_Design]. For new Semantic Web languages this reinforces what has been similarly known in the

Lisp community for decades – that an XML markup and a concise syntax should be co-designed with a pair of translators permitting smooth transitions between the two. This paper is based on the pair RuleML↔POSL, whose evolving components are supported by translators [http://www.ruleml.org/posl/converter.jnlp] for combining 'deep' (XML) markup with a 'shallow' (ASCII) shorthand, online. These being in place, we can look at the design issues below.

Knowledge representation (KR) languages have been developed, with limited time and cross-fertilization, to cover the following Semantic Web design space: Object-centered resource instance descriptions via binary properties (RDF), taxonomies over resource classes and properties (RDFS), description logic with class-forming operations and class/property axioms (OWL), as well as derivation, integrity, transformation, and reaction rules (RuleML). At the bottom of, or combined with, these languages, different kinds of (binary, n-ary) ground facts have been used besides database tuples for representing instances. On top of, or again combined with, these languages, query languages have been defined. Integrations of various of these languages have been developed, including the combination of object-centered descriptions and rules (N3, OO RuleML) as well as description logic and rules (Description Logic Programs, SWRL). These language integrations can help with information integration on the Web such as mapping object-centered representations to positional ones. The POSL research has explored this design space and introduced orthogonal ('decoupled') dimensions for systematic Semantic Web language development. The orthogonal design has allowed to incorporate most of the above notions in such a way that they can be used and revised independently from each other.

Two language families that predated the (Semantic) Web, yet have been very useful for it, are positional languages based on Horn logic such as Prolog, and slotted languages with object-centered instance and class descriptions plus rules as in F-logic [KL89,YK03]. Both have concise ASCII syntaxes, elegant semantics, and decent computational properties. Since these positional and slotted styles are often needed conjointly in the XML&RDF Web, they have been integrated in

POSL. Prolog and F-logic can be given additional XML syntaxes, and adapted to the Semantic Web by webizing key language elements via URIs as well as permitting modular and distributed knowledge bases. In POSL, the integrated XML syntax is OO RuleML, and integrated webizing is done in the style of N3 for all language elements. The POSL syntax is given in appendix A.

Based on projects at NRC and UNB, we will exemplify all POSL notions through an e-Business use case in logistics. Without the POSL layer available above the XML level, it would not have been possible to complete, in the allotted time, the development cycles of recent Web applications such as RACOFI Music [ABB+03] [http://racofi.elg.ca/about.html], the New Brunswick Business Knowledge Base [MB05] [http://www.ruleml.org/usecases/nbbizkb], and FindXpRT [LBBM06] [http://www.ruleml.org/usecases/foaf].

2   The Positional-Slotted Space for Facts and Queries

Positional and slotted KR both have advantages for various tasks in the Semantic Web, hence in POSL are reconstructed in a space with orthogonal dimensions, also allowing for various mixed and extended forms:

	ordered	*unordered*
keyed	notched	**slotted**
unkeyed	**positional**	collectional

The arguments of n-ary relations and objects can, independently, be *ordered* or *unordered*, *keyed* or *unkeyed*. The 'main diagonal' combinations, *positional* = ordered + unkeyed and *slotted* = unordered + keyed, are more common, hence will be focused here; the 'secondary diagonal' combinations, *collectional* = unordered + unkeyed and *notched* = ordered + keyed, will also be useful.

Beginning with our logistics use case, a 4-ary relation `shipment` can represent the shipping of some cargo at a price from a source to a destination. The corresponding `shipment` relationships (atoms) can be represented in all

notations discussed in section 1, hence are used to illustrate the POSL space.

Positional notations have been used, intuitively, for ordered sequences of possibly repeating objects; in mathematics (hence in physics, chemistry, etc.), for n-tuples and for the arguments to n-ary functions; in logics, for the arguments to n-ary relations, in programming, database, and KR languages; for n-ary functions and relations, in XML, for child elements within a parent element; as well as in RDF, for `Sequence` containers with ordered `rdf:li` children.

For example, the `shipment` relation can use the cargo, price, source, and destination directly as arguments, in that order. Corresponding `shipment` atoms can be represented as in relational database tuples, Datalog facts, etc.

For this, POSL uses a Prolog-like syntax, e.g. obtaining the following two ground facts (constants may be symbols, with a lower-case or upper-case first letter, or numbers):

```
shipment(PC,47.5,BostonMoS,LondonSciM).
shipment(PDA,9.5,LondonSciM,BostonMoS).
```

Slotted notations have been used, intuitively, for unordered sets of attribute-value pairs; in mathematics, for labeled graphs and finite maps; in frame logics, for molecular formulas; in programming, database, and KR languages, for records and object-centered relations; in XML, for attributes within start tags, as well as in RDF, for resource descriptions via properties.

For example, the above positional 4-ary `shipment` relation can also be conceived in a slotted manner, where *slot names* such as `cargo` identify the roles of the arguments so that their order becomes irrelevant. Corresponding `shipment` atoms can then be represented as object-oriented database nodes, frame logic facts, etc. by pairing slot names such as `cargo`, `price`, `source`, and `dest`(ination) with their *slot fillers* such as `PC`, `47.5`, `BostonMoS`, and `LondonSciM`, respectively.

For this, POSL uses an F-logic-inspired syntax, now obtaining these facts ("*name->filler*" slots are separated by a ";" infix, indicating unorderedness):

```
shipment(cargo->PC;price->47.5;source->BostonMoS;dest->LondonSciM).
```

```
shipment(cargo->PDA;price->9.5;source->LondonSciM;dest->BostonMoS).
```

Positional-slotted notations have also been used combined, e.g., in Lisp, for obtaining the benefits of both KR methods.

For example, the above 4-ary `shipment` relations can also be split and re-combined in a positional-slotted manner, where a positional part is followed by a slotted part. The first two arguments, `cargo` and `price`, are rather self-explaining in the positional notation, but the last two arguments, `source` and `dest`ination, could be confused, hence are given here explicit slot names in the combined positional-slotted notation.

For this, POSL uses a Prolog/F-logic-combining syntax, obtaining these facts (the "," infix has precedence over the ";" infix):

```
shipment(PC,47.5;source->BostonMoS;dest->LondonSciM).
shipment(PDA,9.5;source->LondonSciM;dest->BostonMoS).
```

Existing relations such as `shipment` may require later enhancement by further information such as `start` and `duration`. Instead of extending the argument sequences of all affected positional or positional-slotted atoms by further proper and null-valued arguments in an ordered, positional manner, it is often preferable to add only non-null arguments in an unordered, slotted manner.

Structures and plexes go beyond the Datalog language considered so far. All three notations are also possible for any (Prolog-like) structure, e.g. describing a pair of stakeholders as follows (using "[...]" for constructor applications): the positional `stakepair[PeterMiller,SpeedShip]`, the slotted `stakepair[owner->PeterMiller;shipper->SpeedShip]`, and the positional-slotted `stakepair[,PeterMiller;shipper->SpeedShip]`, employing a singleton positional subsequence `,PeterMiller` (for ε`,PeterMiller` with "ε" representing the empty sequence). Similarly, a *plex* is regarded as the special case of a constructorless structure, e.g. changing our stakeholder pairs thus: the Prolog-like list `[PeterMiller,SpeedShip]`, the F-logic/Lisp-inspired association list `[owner->PeterMiller;shipper->SpeedShip]`, and the combination

```
[,PeterMiller;shipper->SpeedShip].
```

Non-ground formulas contain at least one variable argument, interpreted as universally quantified in facts and as existentially quantified in queries. They are allowed for all three notations. However, variables are not permitted as slot names in (First-Order) POSL since there would no longer be a unique most general unifier, so non-determinism would already arise during the unification phase of resolution. Variables can be named or anonymous. Named variables are prefixed by a "?"; the anonymous variable is written as a stand-alone "?". For example, for the earlier positional `PC-shipment` ground fact, the non-ground query `shipment(PC,?,BostonMoS,?goal)` succeeds, unifying the anonymous "?" with `47.5` and binding `?goal` to `LondonSciM`.

Rest arguments are permitted in atoms, one for positional arguments and one for slotted arguments. Positional arguments are separated from a positional rest by a "|"; slotted arguments are separated from a slotted rest by a "!". In both cases the rest itself is normally a variable, enabling a varying number of arguments, thus making an atom *polyadic* – the fixed-arity/polyadic distinction being orthogonal to the positional/slotted distinction. In particular, the anonymous variable can be used as a positional or slotted "don't care" rest. A slotted "don't care" rest "!?" makes an option from F-logic's fixed convention: to tolerate arbitrary *excess slots* in either formula (e.g., a fact), having slot names not used by any slot of the other ("!?"-)formula (e.g., a query), for unification.

For example, for the earlier slotted `PC-shipment` fact, the query

```
shipment(cargo->?what;price->?;source->BostonMoS;dest->?goal)
```

succeeds, binding `?what` to `PC` and `?goal` to `LondonSciM`. However, the query

```
shipment(owner->?who;cargo->?;price->?;source->BostonMoS;dest->?)
```

fails because of its excess slot named `owner`. Similarly, the query

```
shipment(cargo->?what;source->BostonMoS;dest->?goal)
```

fails because of the fact's excess slot named `price`. On the other hand, the query with the slotted "rest doesn't care" combination "!?"

```
shipment(cargo->?what;source->BostonMoS;dest->?goal!?)
```

again succeeds with the initial bindings, since "!?" anonymously unifies the `price` slot (independent of where it occurs in the fact).

Conversely, the earlier fact would tolerate excess query slots such as in the above `owner` query after 'opening it up', non-ground, via an anonymous rest:

```
shipment(cargo->PC;price->47.5;
         source->BostonMoS;dest->LondonSciM!?).
```

If the query also contains an anonymous rest, both it and the fact can contain excess slots, as in

```
shipment(owner->?who;cargo->?what;source->BostonMoS;dest->?goal!?)
```

which succeeds with the initial bindings, since the anonymous query rest unifies the fact's `price` slot and the anonymous fact rest unifies the query's `owner` slot, leaving the variable `?who` free, and the querier agnostic about the owner.

If anonymous rest slots are employed in all formulas, the effect of F-logic's implicit rest variables is obtained. The more precise, "!"-free slotted formulas can enforce more restricted, 'closed-off' unifications where needed.

In general, "|" and "!" rests can follow after zero or more fixed positional and slotted arguments, and can unify the zero or more remaining arguments. Before being bound to a variable, such a polyadic rest $e_1, \ldots, e_Z$ or $s_1 \rightarrow f_1; \ldots; s_Z \rightarrow f_Z$ is made into a single plex `[`$e_1, \ldots, e_Z$`]` or `[`s_1`->`f_1`;` $\ldots$ `;`s_Z`->`f_Z`]`, respectively.

Using both kinds of rests, we give below the most general forms of ordered/unordered, keyed/unkeyed atoms (1) and structures (2). Here, the o_i and u_i arguments are ordered and unordered, respectively, and can either be keyed (having the form s_i`->`f_i) or unkeyed (having any other form). The positional-slotted forms are the common special case where exactly the u_i arguments are keyed and exactly the o_i arguments are unkeyed. The equation right-hand sides

show normal forms with all unordered arguments to the right of all ordered arguments (for positional-slotted, all slots to the right of all positionals):

$$r(u_1; ...; u_L; o_1, ..., o_M|V_o; u_{L+1}; ...; u_N!V_u) = r(o_1, ..., o_M|V_o; u_1; ...; u_N!V_u) \quad (1)$$

$$c[u_1; ...; u_L; o_1, ..., o_M|V_o; u_{L+1}; ...; u_N!V_u] = c[o_1, ..., o_M|V_o; u_1; ...; u_N!V_u] \quad (2)$$

The **semantics** of POSL clause sets will be based on slotted (positional-slotted etc.) extensions to the positional (here, LP [Llo87]) notions of clause instantiation and ground equality (for the model-theoretic semantics) as well as unification (for the proof-theoretic semantics).

With slot names assumed to be non-variable symbols, *slotted instantiation* can recursively walk through the fillers of slots, replacing any variables encountered with their dereferenced values from the substitution (environment).

Since POSL uses no implicit rest variables, *slotted ground equality* can recursively compare two ground atoms or structures after lexicographic sorting – w.r.t. the slot names – of the slotted elements encountered.

Since POSL uses at most (one positional and) one slotted rest variable on each level of an atom or structure, *slotted unification* can perform sorting as in the above slotted ground equality, use the above slotted instantiation of variables, and otherwise proceed left-to-right as for positional unification, but pairing up identical slot names before recursively unifying their fillers, while collecting excess slots on each level in the plex value of the corresponding slotted rest variable.

3 Horn-like Rules Typed via RDFS or OWL Classes

On top of positional and slotted facts, and in the same integrated manner, POSL offers Horn-like rules for inferential tasks in the Semantic Web. Facts are interpretable as clauses that are degenerated (premiseless) rules, which in POSL can be naturally extended to clauses that are full-blown (premiseful) rules.

Extending the logistics use case, a ternary relation `reciship` can represent reciprocal shippings of unspecified cargos at a total cost between two sites. A

Datalog rule infers this conclusion from three premises, two `shipment` atoms and an `add` atom. The `shipment` relation was defined in section 2 and the `add` relation is based on a SWRL built-in satisfied here iff the first argument is equal to the sum of the second and third arguments.

Positional rules are the usual Horn rules, in POSL written using a Prolog-like syntax, but again employing "?"(-prefixed) variables as, e.g., in Jess, N3, and Common Logic. In the `reciship` example, the following Datalog rule is obtained (the ":-" infix, for "⇐", has lowest precedence):

```
reciship(?cost,?A,?B) :-
  shipment(?,?cost1,?A,?B),
  shipment(?,?cost2,?B,?A),
  add(?cost,?cost1,?cost2).
```

The query `reciship(?total,BostonMoS,LondonSciM)` uses the rule to itself query the corresponding `shipment` facts and call the `add` built-in, binding `?total` to `57.0`. This query could be chained to from the body of another positional rule, e.g., `reciBosLon(?total) :- reciship(?total,BostonMoS,LondonSciM)`.

With types `Float` and `Address` for the head as well as additional `Product` and `Float` types for the extra anonymous and `?cost`-named body variables, defined as RDFS or OWL classes (e.g., as in section 5), the above `reciship` rule becomes fully typed as follows (we use the classical ":" infix between a variable and its type, which, applied once, holds for all of its occurrences in a clause):

```
reciship(?cost:Float,?A:Address,?B:Address) :-
  shipment(?:Product,?cost1:Float,?A,?B),
  shipment(?:Product,?cost2:Float,?B,?A),
  add(?cost,?cost1,?cost2).
```

Slotted rules are much like in F-logic. The `reciship` relation is redefined here in a slotted manner with slot names `price`, `site1`, and `site2`, where two 'indexed' `site` slots are used. Analogously, the positional `add` relation could be made slotted via extra slot names `sum`, `addend1`, and `addend2`:

```
reciship(price->?cost;site1->?A;site2->?B) :-
  shipment(cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Notice that the slot name `price` occurs here both in the relation `shipment`, for elementary costs, and in the relation `reciship`, for an aggregated cost. Similarly, while the `dest` slot in the `shipment` relation is of type `Address`, a slot with the same name in a `flight` relation could have type `AirportCode`. Such 'overloading' is caused by slot names, except when 'webized' (cf. section 5), being local to their relations much like property restrictions are local to their class descriptions in OWL [DS04].

Now, `reciship(site1->BostonMoS;price->?total;site2->LondonSciM)` through `reciship(price->?total;site1->BostonMoS;site2->LondonSciM)`, the lexicographically sorted normal form, queries the slotted rule, which itself queries corresponding clauses, again binding `?total` to `57.0`. The original query could be chained to from the body of another slotted rule, e.g. having the head `reciBosLon(price->?total)`.

The above rule can again use variable typing:

```
reciship(price->?cost:Float;site1->?A:Address;site2->?B:Address)
  :-
  shipment
    (cargo->?:Product;price->?cost1:Float;source->?A;dest->?B),
  shipment
    (cargo->?:Product;price->?cost2:Float;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Positional-slotted rules use at least one positional and one slotted relation as the conclusion or some of the premises, or use at least one positional-slotted relation as the conclusion or some of the premises. For instance, to avoid the 'indexed' slot conventions/assumptions in the slotted rule above, a positional-

slotted rule can be positional for the conclusion and the `add` premise, and can be slotted for the `shipment` premises:

```
reciship(?cost,?A,?B) :-
  shipment(cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(cargo->?;price->?cost2;source->?B;dest->?A),
  add(?cost,?cost1,?cost2).
```

The **semantics** of slotted and positional-slotted clause sets can be defined on top of the semantic basis for atoms and structures in section 2. Since on the level of clauses all three notations have the same interpretation, the treatment in section 2 naturally extends to slotted (and positional-slotted) generalizations of positional (LP [Llo87]) clauses. The further semantic treatment via Herbrand models and resolution proof theory directly follows the one for the positional notation [Llo87]. The semantics of typing (sorts) could be given directly but can also be reduced to the unsorted case in a well-known manner: All occurrences of a sorted variable are replaced by their unsorted counterparts plus a body-side application of a sort-corresponding unary predicate to that variable (sorted facts thus become unsorted rules); moreover, the definition of the unary predicate reflects the subsumption relations of the sort taxonomy via rules.

The **implementation** of POSL for slotted and positional-slotted clauses, called OO jDREW, has followed the semantics via an extension of the Java-based jDREW interpreter [BBH+05]; it is available via Java Web Start and for full download [http://www.jdrew.org/oojdrew]. In sorted Prologs the implementation of typing was performed directly (without the above reduction) before RDFS and OWL became available as Web-based taxonomy languages. We have begun to adapt sorted indexing techniques to RDFS and to the OO jDREW interpreter for POSL.

Several **applications** of POSL have used its OO jDREW implementation, e.g. rules for information integration in the New Brunswick Business Knowledge Base [MB05] and for expert finding in FindXpRT [LBBM06].

4 Signatures

POSL uses optional signature declarations, particularly to help with knowledge base integration in Web-distributed development. Signatures can equip arguments with slots and types, which, as shown in section 5, may refer to classes defined in a Web taxonomy language such as RDFS or OWL-DL.

A signature declaration has the form of a fact except that "*" instead of a "." is used as the terminator. For any relation, zero or more signature declarations are permitted, which conjointly constrain the relation's applicability.

For our positional facts, a signature can be declared to specify their arity (implicitly, 4) and argument types ("?:" is used as a type 'prefix') as follows:

```
shipment(?:Product,?:Float,?:Address,?:Address)*
```

For the slotted and positional-slotted facts, signatures can be declared thus (all or some "," infixes are replaced by ";"):

```
shipment(cargo->?:Product;          shipment(?:Product,
        price->?:Float;                     ?:Float;
        source->?:Address;                  source->?:Address;
        dest->?:Address)*                   dest->?:Address
                                            !?)*
```

The left-hand, slotted signature gives slots and filler types to all of its arguments (no excess slots will be tolerated because of the absence of "!?"). The right-hand, positional-slotted signature specifies its positional as well as its slotted arguments (without "|?" not tolerating excess positional arguments but with "!?" tolerating excess slots). In both signatures the same type ?:Address now occurs in two differently named 'roles', for the source and dest slots.

Signature declarations are also allowed for a relation defined by a clause set containing rules, for whose heads they again specify slot names and argument types, as for facts. Signatures that themselves have the form of rules, where all signature rules of a relation must unify and succeed, are currently not allowed.

The **semantics** of POSL signatures is that of filters over a candidate model's ground facts having the same relation name: Only ground facts unifying, order-sorted, with all of their signatures will stay in the model.

5   Webizing Individuals, Relations, Slots, And Types

The POSL language elements of individuals (and constructors), relations, slots, and types can be webized, and generally can be given URIs. Different occurrences of the same language element can thus be disambiguated by giving them different URIs. Since it concerns language elements wherever they occur, POSL webizing is orthogonal to the positional/slotted distinction.

First, we distinguish two kinds of character sequences that have the form of URIs in the POSL KR language: An active URI, meant to identify a resource (the usual case), is enclosed in a pair of angular brackets, `<...>`, following IETF's generic URI syntax [http://gbiv.com/protocols/uri/rev-2002/rfc2396bis.html] and N3 [http://www.w3.org/2000/10/swap/Primer]; a passive URI, meant to stand for itself as a string (the unusual case), is enclosed in a pair of double quotes, `"..."`, exactly as other strings in POSL or in other languages. XML namespace prefixes and local names as well as general QNames can then be expressed via variables bound to active URIs (although XML applications like XSLT and RDF use `"..."` or even `'...'` for what is here called active URIs). While whitespace (e.g., any line-break) is ignored in (e.g., long) active URIs, it of course counts in strings.

A symbolic POSL language element occurrence can be associated with an active URI via symbol-URI juxtaposition, generalizing a wide-spread convention for user-email association as in `"Fred Bird"<mailto:sales@sphip.com>`. A POSL element such as the string individual `"Fred Bird"` can also be entirely replaced by a URI, as in the stand-alone `<mailto:sales@sphip.com>`.

Webized individuals employ active URIs in place of, or in addition to, individual-constant symbols. For example, `SpeedShip` can be associated

with an active URI for the intended speed shipping company's homepage `<http://sphip.com>` to obtain the following webized individual:

```
SpeedShip<http://sphip.com>
```

Our 4-ary positional `shipment` fact from section 2 can now be extended by a shipping company as the first argument of a 5-ary fact using one of three options.

(1) The individual symbol `SpeedShip` itself can be used, as we did with `BostonMoS` etc. before webizing:

```
shipment(SpeedShip,PC,47.5,BostonMoS,LondonSciM).
```

(2) The active URI can be employed in place of the individual symbol, as practiced in RDF, N3, and other Web KR languages:

```
shipment(<http://sphip.com>,PC,47.5,BostonMoS,LondonSciM).
```

(3) The webized individual symbol can be employed, as defined in RuleML:

```
shipment(SpeedShip<http://sphip.com>,PC,47.5,BostonMoS,LondonSciM).
```

The same options exist for slotted facts, as exemplified with the most general option (3), enriched by webized `BostonMoS` and `LondonSciM` individuals:

```
shipment(shipper->SpeedShip<http://sphip.com>;
         cargo->PC;
         price->47.5;
         source->BostonMoS<http://www.mos.org/info/contact.html>;
         dest->LondonSciM<http://www.sciencemuseum.org.uk/
                          visitors/location.asp>).
```

Notice that the new positional first argument caused all former arguments to shift by one, while the new slotted argument was added without affecting the interpretations of the existing slotted or any positional arguments (thus better supporting argument inheritance and distributed knowledge development).

Webized relations employ active URIs in place of, or in addition to, symbolic relation names. For example, the 4-ary and 5-ary positional `shipment` relations can be uniquely distinguished via URIs pointing to different signatures:

```
shipment<http://transport.org/rels/pos/shipment#4>
shipment<http://transport.org/rels/pos/shipment#5>
```

These webized relations can now be used unambiguously as follows (the `shipment` symbol in front of the URIs could be omitted):

```
shipment<http://transport.org/rels/pos/shipment#4>
                        (PDA,9.5,LondonSciM,BostonMoS).
shipment<http://transport.org/rels/pos/shipment#5>
                        (SpeedShip,PC,47.5,BostonMoS,LondonSciM).
```

Similarly, 4-ary, 5-ary, and polyadic slotted `shipment` relations could be distinguished via URIs pointing to different signatures or, for the latter case, to RDFS-like `subPropertyOf` information (polyadicity is represented by an "`X`"):

```
<http://transport.org/rels/slot/shipment#4>
<http://transport.org/rels/slot/shipment#5>
<http://transport.org/rels/slot/shipment#X>
```

Sample uses will be demonstrated in section 6.2.

Webized slots employ active URIs in place of, as pioneered by RDF, or in addition to, symbolic slot names. For example, the `shipment` slots may be drawn from URIs containing fragmentid's with the original slot names, except for the `charge` fragmentid, for which the local slot name `price` is kept:

```
shipment(<http://transport.org/slots/shipment#shipper>->SpeedShip;
        <http://transport.org/slots/shipment#cargo>->PC;
        price<http://ebizguide.org/slots#charge>->47.5;
```

```
          <http://trajectory.org/slots/movement#source>->BostonMoS;
          <http://trajectory.org/slots/movement#dest>->LondonSciM).
```

Webized types use a URI reference to an RDFS or OWL class. For example, the `Product` type can be associated with a URI for the corresponding OWL class:

```
Product<http://www.daml.org/services/owl-s/1.0/
        ProfileHierarchy.owl#Product>
```

Using this for typing the anonymous variable of our positional rule in section 3, a primitive from XML Schema Datatypes for its `cost`-like variables, and a webized `Address` type, we obtain the following Web-typed rule:

```
reciship(?cost:Float<http://www.w3.org/TR/2001/
                     REC-xmlschema-2-20010502/#float>,
        ?A:<http://ebizguide.org/types#Address>,
        ?B:<http://ebizguide.org/types#Address>) :-
  shipment(?:Product<http://www.daml.org/services/owl-s/1.0/
                     ProfileHierarchy.owl#Product>,
         ?cost1:Float<http://www.w3.org/TR/2001/
                      REC-xmlschema-2-20010502/#float>,
         ?A,?B),
  ... .
```

A semantics of webizing, for URI grounding (or anchoring), has been based on a notion of URI equality via string rewriting for normalization [Bol03].

6 Anchored Atoms for OO Knowledge Representation

Webizing is also possible for entire atoms, as a way of associating them with Object IDentifiers (OIDs). Generally, fact atoms can be *anchored* by an OID (a symbolic name or an active URI, possibly prefixed by a symbolic name) as a special 'zeroth' argument separated from further arguments by an up-arrow/hat

infix "^": $relation(oid\,^\,arg_1...arg_N)$. Anchoring uniformly extends relations to objects.

For example, the earlier 4-ary positional and slotted facts (see "%" comments) can now be anchored using variously webized versions of names like s1 and s2:

```
shipment(s1^PC,47.5,BostonMoS,LondonSciM).           % positional
shipment(<http://sphip.com/event#s2>^PDA,9.5,LondonSciM,BostonMoS).

shipment(s1^cargo->PC;price->47.5;                   % slotted
         source->BostonMoS;dest->LondonSciM).   % (*)
shipment(s2<http://sphip.com/event#s2>^...).
```

In the same way, rule head and body atoms can be webized, e.g. for deriving and querying specifically identified facts.

For example, the positional and slotted rules from section 3 can now be anchored using versions of the name r1 for the aggregated shipping cost derivation from the queried reciprocal s1 and s2 facts:

```
reciship(<http://sphip.com/rule#r1>^?cost,?A,?B) :-  % positional
  shipment(s1^?,?cost1,?A,?B),
  shipment(s2^?,?cost2,?B,?A),
  add(?cost,?cost1,?cost2).

reciship(r1^price->?cost;site1->?A;site2->?B) :-     % slotted
  shipment(s1<http://sphip.com/event#s1>^          % (**)
           cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(s2^cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

6.1   F-logic Objects, Nestings, And Restricted ∧-Composition

Anchored, slotted facts correspond to *object descriptions* in F-logic, where POSL relations correspond to F-logic classes. For example, the above slotted s1 fact (*) corresponds to this F-logic object:

```
s1[cargo->PC,price->47.5,
   source->BostonMoS,dest->LondonSciM]:shipment.
```

Notice that POSL puts the relation name, `shipment`, in front of parentheses, as in conventional relational notation, extended with the Object IDentifier, `s1`, in a special argument position, while F-logic object descriptions put the OID in front of brackets, ":"-separated from the class name (F-logic signatures again put the class in front of the brackets).

F-logic's nesting shorthand for object descriptions is reflected by n-ary anchored POSL facts through the following left-to-right-normalizing equations:

$$r(oid_r\hat{}...; s\text{->}q(oid_q\hat{}\ldots);...). \; = \; r(oid_r\hat{}...; s\text{->}oid_q;...). \; q(oid_q\hat{}\ldots). \quad (3)$$

For example, the `PC` of our `s1` fact (*) can be defined as an embedded object with an OID `s3` carrying its own `value` and `weight` slots:

```
shipment(s1^cargo->PC(s3^value->2500.0;weight->17.5);price->47.5;
             source->BostonMoS;dest->LondonSciM).
```

According to transformation (3), this shorthand normalizes to two anchored, slotted facts:

```
shipment(s1^cargo->s3;price->47.5;
             source->BostonMoS;dest->LondonSciM).
PC(s3^value->2500.0;weight->17.5).
```

While the nested version defines the `s3` object within the `s1` object, the unnested version clarifies that the OID `s3` is on the same definition level as `s1`. Since this allows (possibly undesired) external access to such an OID, section 6.2 will show how it can be localized as an anonymous/blank node.

F-logic's ∧-composition shorthand in POSL is restricted to (non-empty) *independent* groups of object slots only, i.e. an anchored rule is decomposable if

it can be partitioned (without loss of generality, after possible reordering of its slots and body premises) into subrules that do not share variables in the head, the body, or cross-wise:

$$r(oid\verb|^|s_1\verb|->|f_1; ...; s_i\verb|->|f_i; s_{i+1}\verb|->|f_{i+1}; ...; s_N\verb|->|f_N) \; \verb|:-| \; b_1, ..., b_j, b_{j+1}, ..., b_P . \quad = \quad (4)$$

$$r(oid\verb|^|s_1\verb|->|f_1; ...; s_i\verb|->|f_i) \; \verb|:-| \; b_1, ..., b_j .$$

$$r(oid\verb|^|s_{i+1}\verb|->|f_{i+1}; ...; s_N\verb|->|f_N) \; \verb|:-| \; b_{j+1}, ..., b_P .$$

$$\text{if } 1 \leq i \leq N-1 \; \wedge \; 0 \leq j \leq P$$

$$\wedge \; \mathrm{vars}(\{f_1, ..., f_i\}) \cap \mathrm{vars}(\{f_{i+1}, ..., f_N\}) = \{\}$$

$$\wedge \; \mathrm{vars}(\{b_1, ..., b_j\}) \cap \mathrm{vars}(\{b_{j+1}, ..., b_P\}) = \{\}$$

$$\wedge \; \mathrm{vars}(\{f_1, ..., f_i\}) \cap \mathrm{vars}(\{b_{j+1}, ..., b_P\}) = \{\}$$

$$\wedge \; \mathrm{vars}(\{f_{i+1}, ..., f_N\}) \cap \mathrm{vars}(\{b_1, ..., b_j\}) = \{\}$$

This restriction is similar to the one used for independent $\wedge$-parallelism [HR95], which for the Web's distributed object definitions captures those parts of objects that can be defined independently from other parts. POSL's *independent $\wedge$-decomposition* permits maximum distribution of rule-defined objects and seems to resolve an issue with F-logic's frame notation mentioned in recent SWSL discussions [Kif05].

For example, the slotted `s1` fact (*) is ground, hence is fully decomposable into four facts using three applications of (4), exactly reflecting F-logic's shorthand (the decomposed facts can be $\wedge$-connected within a rulebase or, if `s1` is globally unique, across distributed rulebases):

```
shipment(s1^cargo->PC).
shipment(s1^price->47.5).
shipment(s1^source->BostonMoS).
shipment(s1^dest->LondonSciM).
```

On the other hand, this `s4` fact is non-ground and two slots share a variable:

```
sightseeingflight(s4^passenger->?x;price->100;source->?z;dest->?z).
```

Hence, `s4` is decomposable only in a restricted manner. Maximally two applications of (4) produce three facts:

```
sightseeingflight(s4^passenger->?x).
sightseeingflight(s4^price->100).
sightseeingflight(s4^source->?z;dest->?z).
```

The following refinement into an `s5` rule makes the other two slots dependent, with co-occurring variables in a relation call:

```
sightseeingflight(s5^passenger->?x;price->?y;source->?z;dest->?z)
  :-
  ticket(?pronumber^passenger->?x;price->?y).
```

Therefore, `s5` is maximally decomposable into just two clauses with a single application of (4), where the only body premise is kept for the first clause, while the (implicitly true) empty body is given to the second clause:

```
sightseeingflight(s5^passenger->?x;price->?y) :-
  ticket(?pronumber^passenger->?x;price->?y).
sightseeingflight(s5^source->?z;dest->?z).
```

Finally, the above slotted `reciship` rule (`**`) is not decomposable by (4) at all, since there is no variable-disjoint partition of its body calls and every head variable also occurs in the body.

6.2 RDF Descriptions, Blank Nodes, And Rules

RDF descriptions can then be conceived as anchored slotted facts, in the absence of `rdf:type` using the null relation.

If we assume that our 5-ary slotted fact in section 5, by virtue of the `shipper` slot and other ones, can only be a *shipping* relationship, we may omit an `rdf:type` for shipments, obtaining the following RDF:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:s="http://transport.org/slots/shipment#"
 xmlns:p="http://ebizguide.org/slots#"
 xmlns:m="http://trajectory.org/slots/movement#">
 <rdf:Description about="http://sphip.com/event#s1">
  <s:shipper rdf:resource="http://sphip.com"/>
  <s:cargo>PC</s:cargo>
  <p:charge>47.5</p:charge>
  <m:source rdf:resource="http://www.mos.org/info/contact.html"/>
  <m:dest rdf:resource="http://www.sciencemuseum...location.asp"/>
 </rdf:Description>
</rdf:RDF>
```

This can be represented as a corresponding POSL fact without a relation name, using webizing also for slot names and individuals (but not using its analog to namespace prefixes, explained in the online POSL document [http://www.ruleml.org/submission/ruleml-shortation.html]):

```
(<http://sphip.com/event#s1>^
 <http://transport.org/slots/shipment#shipper>->
                       <http://sphip.com>;
 <http://transport.org/slots/shipment#cargo>->PC;
 <http://ebizguide.org/slots#charge>->47.5;
 <http://trajectory.org/slots/movement#source>->
                       <http://www.mos.org/info/contact.html>;
 <http://trajectory.org/slots/movement#dest>->
                       <http://www.sciencemuseum...location.asp>).
```

Symbolic and webized individuals are represented in the same manner here, so that a symbolic name like PC can later be replaced by a blank node or a URI for its product catalog entry, without changing anything about the enclosing slot.

To increase type-safeness, the webized polyadic relation name from section 5, `<http://transport.org/rels/slot/shipment#X>`, can now be introduced into the above RDF description as an `rdf:type`:

```
<rdf:RDF
  ...
  <rdf:Description about="http://sphip.com/event#s1">
    <rdf:type
      rdf:resource="http://transport.org/rels/slot/shipment#X"/>
    ...
  </rdf:Description>
</rdf:RDF>
```

Such a resource type is considered here as a *relationship type* directly applicable, as a relation name, to the arguments of the corresponding POSL fact:

```
<http://transport.org/rels/slot/shipment#X>
                      (<http://sphip.com/event#s1>^
                       ...).
```

The arity of this POSL fact could be fixed using the webized 5-ary relation name `<http://transport.org/rels/slot/shipment#5>` from section 5 instead.

RDF blank nodes are used for OIDs local to the current document. For example, the earlier shipping description can be refined by referring to a local `cargo` description using the blank node identifier `PeterMillerPC` as follows:

```
<rdf:RDF
  ...
  <rdf:Description about="http://sphip.com/event#s1">
      ...
    <s:cargo rdf:nodeID="PeterMillerPC"/>
    ...
```

```
    </rdf:Description>
    <rdf:Description rdf:nodeID="PeterMillerPC">
      <p:value>2500.0</p:value>
      <p:weight>17.5</p:weight>
    </rdf:Description>
</rdf:RDF>
```

Based on the RDF semantics of [Hay04] and its development in [YK03], this can be represented as the below module of two facts connected by an existential variable, in POSL replaced by a Skolem constant _PeterMillerPC. *Modules*, like N3's contexts, TRIPLE's models, and F-logic's scoped formulas, are enclosed using "{...}", and Skolem constants, whose scope is global to clauses but local to modules, are prefixed by an "_" and usable, e.g., as slot fillers and OIDs:

```
{
  (<http://sphip.com/event#s1>^
   ...
   <http://transport.org/slots/shipment#cargo>->_PeterMillerPC;
   ...).
  (_PeterMillerPC^
   <http://ebizguide.org/slots#value>->2500.0;
   <http://ebizguide.org/slots#weight>->17.5).
}
```

A module can have a constructor, which may be parameterized, TRIPLE-like [SD02]. Across different modules, our Skolem constants, even when equally named, denote different objects. Within any module, our Skolem constants obey a *unique name assumption*: differently named constants denote different objects.

Such module-scoped, *unique Skolem constants* can also be generated by the *New Skolem constant* primitive (written as a stand-alone "_"), where all occurrences "_", "_", ... are semantically replaced by fresh constants _1, _2, ..., skipping any (finite) subsequences of positive integers that are already used as

OIDs in the local module. With the above assumption, all occurrences denote different objects. The model theory for such (New) Skolem constants in rules has been developed on top of an *anonymous-domain-augmented Herbrand universe* by [YK03].

For *anchored unification* (where all Skolem constant occurrences _*skocon* may result from dereferencing "?"-variables), any: _*skocon* succeeds with itself; _*skocon* succeeds with a free variable ?*logvar* (or a stand-alone "?"), binding ?*logvar* to _*skocon*; "_" succeeds with a free variable ?*logvar* (or a stand-alone "?"), binding ?*logvar* to the _*skocon* generated by "_" (rather than to "_" itself).

While the above constructs were introduced for slotted representations with RDF blank nodes, they can be similarly used for positional representations.

RDF-like rules can then be directly defined in POSL to process such facts.

For example, the earlier slotted rule can be modified to query untyped facts, inferring, as new "_"-anchored atoms, the OIDs and aggregated cost of any reciprocal shippings (webized slot names are abridged here using symbolic names):

```
reciship(_^forthtrip->?oid1;backtrip->?oid2;
          price->?cost;site1->?A;site2->?B) :-
  (?oid1^shipper->?;cargo->?;price->?cost1;source->?A;dest->?B),
  (?oid2^shipper->?;cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Notice that the `?oid1`/`?oid2` variables occur in two roles: to the left of "^", as proper OIDs, and to the right of "^", as ordinary data values.

In bottom-up derivations, the "_" generates the next fresh Skolem constant, obtaining facts such as `reciship(_4711^...)`. In top-down queries like `reciship(?obj^...)`, the *OID-request variable* `?obj` is successfully bound to such a fresh Skolem constant. The bottom-up direction is preferable for a (non-Horn) extension with *conjunctive heads* (RDF graphs) sharing Skolem constants.

Such rules can be employed within a semantic search engine operating on RDF/POSL-described metadata for obtaining high-precision results: in the above example, priced pairs of Web objects about A-to-B and B-to-A shippings.

7 Conclusions

This paper introduces a core of positional and slotted notions plus notations for KR on the Semantic Web and e-Business knowledge interchange.

A notion not treated in this paper is negation in POSL, for which negation-as-failure (Naf or "∼"), strong-negation (Neg or "¬") and a combination (Naf of Neg or "∼ ¬") are allowed as in RuleML. These distinctions can again be added to the other POSL distinctions as an orthogonal dimension, and their (stable model) semantics adapted from ERDF [AADW05].

Only relations and their defining Horn clauses have been presented here. However, functions defined by (conditional) equations can be added as in Relfun and Functional RuleML [http://www.ruleml.org/fun].

Current work concerns a general POSL treatment of slot cardinalities. While the F-logic system FLORA-2 distinguishes single-valued from set-valued attributes, the description logic system OWL DL provides exact, min, and max cardinality restrictions. The POSL design as presented in this paper employs single-valued slots. However, our plex data with only collectional elements constitute bags (finite multisets), which can represent fillers of multiple-valued slots.

A topic of future research is the issue of extending OIDs towards a general notion of object identity. Actually, there can be several (M) objects to the left of the POSL "^" infix, targeted by an (N-ary) operation: $operation(oid_1...oid_M \hat{} arg_1...arg_N)$. This can provide a bridge from the declarative OO KR rules studied here to OOP-like reaction rules and Web Services. For example, with $M=2$ and $N=1$, the message `transfer(checking1,savings2^,3500)` addresses equally focussed account objects `checking1` and `savings2` in a positional manner, using the single positional argument `3500` for the amount to be transferred in the 'from-to' di-

rection. Besides such ","-ordered receiver objects, also ";"-unordered ones can be used for parallel message broadcasting. For example, with $M{=}2$ and $N{=}2$, the message `equalize(checking1;checking2^min->1000;max->2000)` addresses equally focussed objects `checking1` and `checking2` in an unordered manner, using slotted arguments for the minimal amount, `1000`, and the maximal amount, `2000`, to be left in both accounts after a balancing transfer in either direction, if their amounts were unequal. In practice, such symbolic account names would be replaced by password-protected URIs.

Three recent e-Business applications, in music filtering [ABB$^+$03], information integration [MB05], and expert finding [LBBM06] were enabled by POSL technology. F-logic, SWSL, WRL, SWRL, RIF, and RuleML teams will continue to provide use cases and requirements for the POSL integration presented here.

References

[AADW05] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damásio, and Gerd Wagner. Stable Model Theory for Extended RDF Ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2005.

[ABB$^+$03] Michelle Anderson, Marcel Ball, Harold Boley, Stephen Greene, Nancy Howse, Daniel Lemire, and Sean McGrath. RACOFI: A Rule-Applying Collaborative Filtering System. In *Proc. Collaboration Agents: Autonomous Agents for Collaborative Environments*. Halifax, Canada, October 2003.

[BBB$^+$05] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. Semantic Web Services Language (SWSL). Release Version 1.0, http://www.daml.org/services/swsf/1.0/swsl/, May 2005.

[BBH$^+$05] Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer. Implementing RuleML Using Schemas, Translators, and Bidirectional Interpreters. W3C Workshop on Rule Languages for Interoperability, Position Paper, http://www.w3.org/2004/12/rules-ws/paper/49/, April 2005.

[BK07]   Harold Boley and Michael Kifer. RIF Core Design. W3C Working Draft, W3C, March 2007. http://www.w3.org/TR/2007/WD-rif-core-20070330/.

[Bol03]   Harold Boley. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms. In *Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML-2003)*. LNCS 2876, Springer-Verlag, October 2003.

[DS04]   Mike Dean and Guus Schreiber. OWL Web Ontology Language – Reference. W3C Recommendation, W3C, February 2004.

[GHPM06]   Allen Ginsberg, David Hirtle, Paula-Lavinia Patranjan, and Francis McCabe. RIF Use Cases and Requirements. W3C Working Draft, W3C, July 2006. http://www.w3.org/TR/2006/WD-rif-ucr-20060710/.

[Hay04]   Patrick Hayes. RDF Semantics. W3C Recommendation, February 2004.

[HR95]   M. Hermenegildo and F. Rossi. Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Efficiency, and Compile-Time Conditions. *Journal of Logic Programming*, 22(1):1–45, 1995.

[Kif05]   Michael Kifer. The Relationship Between the Slotted Notation and Frame Notation. Email attachment frames-vs-slots.txt sent to SWSL-Rules team, January 2005.

[KL89]   Michael Kifer and Georg Lausen. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 134–146, Portland, Oregon, 31 May–2 June 1989.

[LBBM06]   Jie Li, Harold Boley, Virendrakumar C. Bhavsar, and Jing Mei. Expert Finding for eCollaboration Using FOAF with RuleML Rules. In *Montreal Conference of eTechnologies 2006*, pages 53–65, 2006.

[Llo87]   John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[MB05]   Anna Maclachlan and Harold Boley. Semantic Web Rules for Business Information. In *Proc. International Conference on Web Technologies, Applications, and Services (WTAS 2005), Calgary, Canada*. IASTED, July 2005.

[SD02]   Michael Sintek and Stefan Decker. TRIPLE – A Query, Inference, and
          Transformation Language for the Semantic Web. In *1st International Se-
          mantic Web Conference (ISWC2002)*. Sardinia, Italy, June 2002.
[YK03]   Guizhen Yang and Michael Kifer. Reasoning about Anonymous Resources
          and Meta Statements on the Semantic Web. In Stefano Spaccapietra, Sal-
          vatore T. March, and Karl Aberer, editors, *J. Data Semantics I*, volume
          2800 of *Lecture Notes in Computer Science*, pages 69–97. Springer, 2003.

A   EBNF Grammar for POSL

```
rulebase  ::= (clause | signature)* .


clause    ::= atom (IMP atoms)? PERIOD .
signature ::= atom ASTERISK .


atoms     ::= atom (COMMA atom)* .
atom      ::= rel LPAREN oid? cont RPAREN .
oid       ::= term HAT .


cont      ::= SEMI? term? | SEMI COMMA | ps .
ps        ::=  (pos prest? | prest) (SEMI set)? srest?
               | set srest pstrail?
               | (set pstrail?)? srest? .
pstrail   ::= (SEMI pos prest? | prest) (SEMI set)? .


prest     ::= PIPE (var | posplex) .
srest     ::= BANG (var | setplex) .


posplex   ::= LBRACK pos? prest? RBRACK .
setplex   ::= LBRACK (SEMI term? | term SEMI set)? srest? RBRACK .


pos       ::= COMMA term? | term (COMMA term)+ .
set       ::= term (SEMI term)* .
```

```
term       ::= slot | unkeyed .
slot       ::= role ARROW unkeyed .


unkeyed    ::=  ind
                | var
                | skolem
                | structure
                | plex .


structure ::= ctor LBRACK cont RBRACK (COLON type)? .
plex       ::= LBRACK cont RBRACK .


ind        ::= (symbol uri? | uri) (COLON type)? .
var        ::= QMARK symbol? (COLON type)? .
skolem     ::= USCORE symbol? (COLON type)? .


ctor ::= rel ::= role ::= type ::= symbol .
```