WORKING PAPER – !! DRAFT, e.g., lacks later sections !!

Version of December 19, 2005

The Production Logic Programs Approach, in a Nutshell: Foundations for Semantically Interoperable Web Rules

Benjamin Grosof
Massachusetts Institute of Technology,
Sloan School of Management,
50 Memorial Drive, Cambridge, MA 02142 USA,
bgrosof@mit.edu, http://ebusiness.mit.edu/bgrosof

Abstract

We overview our foundational work on a knowledge representation (KR) approach of web rules that are semantically interoperable between all four of the currently most commercially important families of rule systems, including production rules, relational database management systems, event-condition-action rules, and Prolog. Called here Production Logic Programs (PLP), the approach combines high degrees of expressiveness, scaleability, and incremental/modular implementability. Its KR is based on declarative logic programs, extending Datalog with several major web-izing and expressive features. These include, notably, a new approach to actions and tests via procedural attachments – in the manner of production rules but with declarative semantics. The PLP approach includes smoothly and powerfully combining rules with ontologies drawn from the currently most commercially important kinds of web-shared semantic ontologies.

The PLP approach has been piloted and evolved in a number of important rule formats, tools, and efforts – including RuleML, SWRL, SWSL/SWSF, WSML/WRL, and SweetRules. An advanced set of PLP capabilities have already been implemented in open source and used to prototype a variety of e-business application scenarios.

Contents

1 Introduction and Overview of the Production Logic Programs Approach

In this paper, we overview our foundational work on a knowledge representation (KR) approach of semantically interoperable web rules. The approach is called here Production Logic Programs (PLP).[1]

PLP combines high degrees of expressiveness, modularity, scaleability, and incremental implementability. The PLP KR is based on declarative logic programs (LP), extending Datalog[2] with several major web-izing and expressive features. The Production LP extension of LP KR is distinguished in particular by adding procedural attachments for actions and tests. It does so in a manner similar to production rules (and event-condition-action rules) – but, unlike them, with a semantics that is disciplined and declarative. This inspires its name.

The PLP approach has a deep semantics that is independent of the inferencing control strategy, algorithms, and implementation – in particular independent of whether rule chaining is forward versus backward in direction. This semantics is defined in terms of what conclusions and triggered actions are sanctioned from a given set of premises. The PLP approach includes bi-directional translations that create powerful semantics-preserving interoperability of inferencing and rulebase merging among all four of the currently most commercially important families of rule systems – production rules, relational database management systems, event-condition-action rules, and Prolog. (The translations are to and from a substantial subset of the expressiveness of each rule system.) The PLP approach has been piloted and evolved in a number of important rule formats, tools, and efforts – including RuleML (Rule Markup Language) [Boley *et al.*, 2005], SWRL (Semantic Web Rule Language) [Horrocks *et al.*, 2004],
SWSL/SWSF (Semantic Web Services Language / Framework) [Battle *et al.*, 2005], WRL/WSML (Web Rule Language / Web Services Modeling Language)[Angele *et al.*, 2005], and SweetRules (toolset) [Grosof *et al.*, 2005], as well as the early IBM CommonRules (toolset) [Grosof *et al.*, 1999a] and EECOMS (supply chain collaboration project) [CIIMPLEX, 1999].

PLP's web-izing features include URI references for predicate etc. ontology, inclusion merging of heterogeneous web-distributed rule and ontology knowledgebases, and markup syntax in XML and RDF.

One set of PLP's expressive features revolve around modularity and nonmonotonicity. This includes, firstly, scoped default negation (a.k.a. well-founded negation-as-failure). It further includes courteous prioritized conflict handling (cf. Courteous extension of LP [Grosof, 2004]), which enables modular and robustly consistent merging/updating of rulebases – a crucial aspect of scaleability for the web.

[1] It supersumes previous work described in terms of the Situated Courteous Logic Programs (SCLP) KR.

[2] more precisely: definite, equality-free, function-free, Horn logic programs

Another set of PLP's expressive extensions revolve around "active-ness" of rules. This includes procedural attachments for actions and tests/queries (similar to, and generalizing, the Situated extension of LP [Grosof, 2004]). It further includes support for incremental event-driven behavior.

The PLP approach includes smoothly and powerfully combining semantic ontologies with rules, so that rules can use web ontologies; rules can also themselves represent web ontologies. In particular, the predicates appearing in rules can be drawn from the currently most commercially important kinds of web-shared semantic ontologies, notably object-oriented (OO) and database schemas. These include RDFS/OWL Description Logic (DL) ontologies [Bechhofer *et al.*, 2004] via the Description Logic Programs (DLP) approach [Grosof *et al.*, 2003] and its extensions, Java/C++/UML style class hierarchies that employ default inheritance, and additional kinds of non-DL restricted First Order Logic (FOL) ontologies.

The PLP approach has attractive computational complexity, qualitatively similar to that of relational databases. It is thus scaleable to very large size knowledge bases (KB's).

Many advanced capabilities of the PLP approach have been already been implemented, via techniques that incrementally and modularly leverage previously existing rule (and ontology) engines and languages.

These advanced capabilities of the PLP approach have been applied to prototype a variety of application scenarios including business and trust policies, advertising and e-contracting, business process monitoring and exception handling, ontological translation and mediation, and intelligent information integration – in security, supply chain, retailing, financial, biomedical, travel, and other domains [Grosof *et al.*, 1999b] [Grosof, 2004] [Grosof and Poon, 2004] [Bhansali and Grosof, 2005] [Li *et al.*, 2003] [Grosof and Dean, 2005].

Reference implementations of a variety of advanced PLP capabilities are available in several open source tools. Many of these were first provided, and then furthermore integrated as a suite, in the SweetRules V2 tool platform [Grosof *et al.*, 2005], initially released in 2004. The Flora-2 system [Kifer and *et al*, 2005] contains a cluster of several other complementary advanced PLP capabilities. The capabilities in SweetRules have been used to demonstrate, for example, effective semantic interoperability between Jess production rules [Friedman-Hill and *et al*, 2004] and XSB Prolog [Warren, 2005] for expressively rich, active, prioritized rules – combined furthermore with ontologies, e.g., in the Description Logic Programs subset [Grosof *et al.*, 2003] of OWL [Bechhofer *et al.*, 2004].

There are a number of further PLP expressive features already developed to some degree and anticipated to be developed further in future work. These include features for Lloyd-Topor enhanced logical connectives and quantifiers [Lloyd, 1987], data types, logical functions, integrity constraints, strong (a.k.a. "classical") negation, Frame (F-Logic) syntax [Kifer *et al.*, 1995] and Hilog higher-order syntax [Chen *et al.*, 1993], skolemization [Yang and Kifer, 2003], explicit equality, and reification

[Battle *et al.*, 2005] (see especially SWSL document section 2). Also already developed to some degree and anticipated to be developed further in future work are expressive features and translations for interoperability with XQuery XML DBMS [Boag *et al.*, 2005], SPARQL RDF DBMS [Prud'hommeaux and Seaborne, 2005], and N3 rule systems (e.g., cwm) [Berners-Lee, 2005].

Many aspects of the Production LP approach have been developed and described in previous papers, presentations, software tools, application scenario prototypes, and emerging standards specifications. In this paper, we focus on describing several of its new aspects that have not been previously described in detail or at all, and on providing a relatively brief overview of the overall approach that incorporates those new aspects.

2 Production LP Extension of LP KR

A new aspect of the Production LP approach, not previously described in papers, is the Production LP extension of declarative LP knowledge representation. In this section, we describe that in a medium level of detail.

A *Production* LP permits action and test expressions, i.e., *situated atoms*, to appear directly within a rule, in a manner fairly similar to production rules.

Actions and tests invoke external procedures, i.e., attached procedures (a.k.a. *aproc's*) that are external to the inferencing process/engine itself. Actions are called for the sake of side effects. Tests are called to query information, and are side-effect-free.

Terminology: The situated literals are said to be "*external*", a.k.a. "*non-local*", since they are expressed in terms of aproc's that are in the external environment. By contrast, the non-situated literals are said to be "*internal*", a.k.a., "*local*". The non-action literals are said to be "*pure-belief*", or more briefly "*pure*", since they represent side-effect-free, truth-valued expressions.

An action or test expression appearing within a rule is an invocation call pattern rather than an invocation itself. An action expression, a.k.a. an *effector atom*, may appear only in the head of a rule. A test expression, a.k.a. a *sensor atom*, may appear only in the body of a rule. During inferencing, an action invocation is triggered after the body of the rule is determined to be satisfied. During inferencing, a test invocation is triggered while the body of the rule is being checked for whether it is satisfied.

For example, consider a simple e-retailing rulebase RB1 consisting of the following rules:

$$\texttt{discountPercent}(?\texttt{customer}, ?\texttt{product}, 25) \\ :- \ \texttt{loyalty}(?\texttt{customer}, \texttt{Gold}) \ \texttt{and} \ \texttt{onSale}(?\texttt{product}).$$

$$@\texttt{sendEmail}(?\texttt{custAddr}, \text{“}\texttt{Cool stuff now on big sale!}\text{”})$$

$$:- \ \texttt{discountPercent}(?\texttt{customer}, ?\texttt{product}, ?\texttt{d})$$
$$\texttt{and prevPurchase}(?\texttt{customer}, ?\texttt{product})$$
$$\texttt{and @customerDBEmailAddrOf}(?\texttt{customer}, ?\texttt{custAddr})$$
$$\texttt{and @greaterThanOrEqual}(?\texttt{d}, 15).$$

together with the following facts:

$\texttt{onSale}(\texttt{SonyDVDPlayer1}).$
$\texttt{loyalty}(\texttt{JaneSmith}, \texttt{Gold}).$
$\texttt{prevPurchase}(\texttt{JaneSmith}, \texttt{SonyDVDPlayer1}).$

Here, "$:-$" is the (usual) LP implication connective, which can be read intuitively as "if" and is sometimes elsewhere written as "$\leftarrow$". The "@" prefix in "$@sendEmail$ syntactically indicates that "$sendEmail$" is the name of an external attached procedure (*aproc*), rather than the name of a predicate. "$sendEmail$" appears in a rule head and is an *action* aproc, a.k.a. an *effector* aproc, intended to be invoked for the sake of its side-effect; invoking it results in the actual sending of an email. Likewise, the "@" prefix indicates that "$customerDBEmailAddrOf$" and "$greaterThanOrEqual$" are also aproc's. However, they appear in a rule body and are *test* aproc's, a.k.a. *sensor* aproc's, intended to be invoked for the sake of querying/obtaining information – without any side effect. Obtaining information via such sensing is similar to accessing a "*virtual*" knowledge base of facts. These facts are not part of the rulebase proper.

Suppose that accessible in this manner via the the sensor aproc's are the (virtual) sensed facts that:

$\texttt{customerDBEMailAddrOf}(\texttt{JaneSmith}, '\texttt{jsmith5555@wahoo61.net}').$
$\texttt{greaterThanOrEqual}(25, 15).$

Then the derived conclusions of the rulebase RB1 include:

$\texttt{discountPercent}(\texttt{JaneSmith}, \texttt{SonyDVDPlayer1}, 25).$

and the following action is sanctioned:

$@\texttt{sendEmail}('\texttt{jsmith5555@wahoo61.net}', ''\texttt{Cool stuff now on big sale!}'').$

Above, for brevity's sake, we did not use webized names (resolving to URI's etc.) for the names of predicates and aproc's and individual constants, and did not give realistically elaborated details in the specification of the aproc's in regard to their interface/signature. Such webized names and aproc details are relatively straightforward to add, for example in the manner of RuleML/SWSL – including its experimental extensions in SweetRules V2.1 – particularly the prefix declarations in [SWSL V1.0 report: section 2.2 "Basic Definitions"] and the presentation/markup syntax for WSDL and Java aproc's in SweetRules V2.1.

In the general case, a sensor/test aproc may require some of its arguments to be bound at the time it is called. A separate kind of statement is used to specify these: a *sensor binding pattern* statement that specifies a binding mode (`FREE` vs. `BOUND`) for each argument/parameter of the sensor aproc. A binding pattern statement is a kind of pragmatic validation constraint. For example:

```
bindreq greaterThanOrEqual (BOUND, BOUND).
bindreq customerDBEmailAddrOf (BOUND, FREE).
```

Here, the first statement specifies that `greaterThanOrEqual` must have both of its arguments bound (i.e., instantiated) at the time it is invoked. The second statement specifies that `customerDBEmailAddrOf` must have its first argument (the customer name) be bound, while its second argument (the customer email address) can be a free variable. This allows the query of `customerDBemailAddrOf` to return the customer email address for a given customer name, not simply to check that a given customer email address corresponds to a given customer name.

Several available binding patterns may be specified for a given sensor aproc, by giving multiple binding pattern statements about it. A call to the sensor aproc must be at least as instantiated as one of the available binding patterns specified (via those statements) for that aproc. This validation requirement is called *br-safety* or *sensor-safety*. For example, adding

```
bindreq customerDBEmailAddrOf (FREE, BOUND).
```

specifies that the customer name can be returned for a given customer email address, as well as vice versa.

Implicitly, every sensor aproc has the *all-bound* binding pattern, in which every argument's binding mode is `BOUND`. All-bound is the most restrictive binding pattern. Conversely, the least restrictive binding pattern is *all-free*, in which every argument's binding mode is `FREE`.

All arguments to an effector aproc must be bound at the time of its call.

Production LP (PLP) generalizes the previous concept of Situated LP. PLP permits also an alternative style of specifying actions and tests, in which an effector (respectively, sensor) statement associates a predicate with an effector (respectively, sensor) aproc. That alternative style was emphasized in most previous work on Situated LP.

Production LP constitutes an expressive extension of LP. It can be combined with many other expressive extensions – i.e., expressive features – of LP in a largely or completely orthogonal (i.e., independent) manner. These other features include scoped default negation (a.k.a. well-founded negation-as-failure), Lloyd-Topor enhanced logical connectives and quantifiers, Courteous, etc.

The **semantics** of the Production extension of LP are defined as a relatively nat-

ural extension of the usual (well-founded) semantics of LP [Van Gelder *et al.*, 1991] [Przymusinski, 1994], without the Production feature.[3]

The fact-form information available via the set of sensor aproc's is viewed as a supplementary virtual knowledge base: the *sensor KB*. That sensor KB has a well founded model that assigns a truth value to each sensor atom.

A truth value is one of $\{true, false, undefined\}$. This kind of *truth value* is as usual in the well-founded semantics of LP. The $false$ truth value means default negation, i.e., intuitively not believed rather than classically false. The $undefined$ truth value indicates an ambiguous status, i.e., intuitively not believed and ambiguous. (The need for, i.e., the presence of, this third truth value $undefined$ arises in certain cases of cyclic dependence through negation that are intuitively ill-defined.)

The sensor KB is merged with a given PLP's rules, and their entailed ground-literal conclusions are drawn in the usual (well-founded) semantic manner to determine the well founded model (WFM) of the PLP. Any rule whose head is an action atom and whose body is satisfied for a given instantiation of the rule's variables generates an entailed – i.e., *sanctioned* – instance of that action corresponding to that instantiation. A given PLP thus derives not only a set of ground conclusion beliefs (as usual for declarative LP), but also a set of sanctioned action calls. The PLP specifies an *episode* of not only pure-belief inferencing, but also of action. The episode is said to be an episode of *inferencing+action*, a.k.a. *situated entailment* or *situated inferencing*. The semantics of a PLP is defined as an abstract function that takes the sensor KB as input and generates a set of sanctioned conclusion beliefs and actions. Formally, the semantics of the PLP consists of:

- a mapping function that, for each WFM of the sensor literals, maps that into a WFM of all the (PLP's) ground literals

The ground literals of the PLP include not only the non-situated literals (as usual in declarative LP) but also the situated literals (effector literals and sensor literals). The well founded model, as usual, consists of a truth value ($true$, $false$, or $undefined$) for each ground literal. The sensor-literals part of the resulting WFM is just the same as the WFM of the input sensor KB.

As part of the definition – and thus semantics – of Production LP, the behavior of aproc's is *disciplined*, i.e., restricted in three regards.

1. *side-effect-free sensors*: As mentioned earlier, sensor aproc's must be side-effect-free; they just return information.

2. *episode-static* aproc's: During a given episode of inferencing/entailment, the timing of sensing and effecting calls does not matter. I.e., one can ignore exactly when, and in what sequence, the sensor and effector calls are invoked during the episode.

[3]Note that it is also possible to define a variant of Production LP semantics in terms of the stable/answer-set style of semantics for LP [Gelfond and Lifschitz, 1988]. We omit details about that here, however.

3. *engine-safe* aproc's: Effector aproc's (side effects) are prohibited from modifying any of: the episode's rulebase premises; the episode's sensor KB; the behavior (including state) of any rule processing engine that is performing the episode's situated entailment/inferencing. I.e., the effector aproc's are *engine-safe*.

This discipline on the procedural attachments ensures that the semantics of Production LP is *declarative* in the KR sense, i.e., independent of the (situated) inferencing control strategy, algorithm, and implementation.

(Another aspect of aproc discipline in PLP, as in production rules typically, is that side-effect-ful aproc's are not called during the checking of rule bodies as they are in Prolog; the behavior of that is highly control-strategy-dependent in Prolog.)

3 Extensible Scope of Definition of "Production Logic Programs"

A second new aspect of the Production LP approach is to define what that is from a KR viewpoint, in a manner similar to how LP is defined from a KR viewpoint. In this section, we do so.

Previous declarative LP theory and practice includes a number of expressive features beyond basic (definite, equality-free) function-free Horn LP. The term (declarative) "logic programs" denotes some KR within that space of generalizations. Many of these generalizations/features – e.g., default negation (a.k.a., negation-as-failure), Lloyd-Topor, Courteous, Situated, Frame, Hilog, integrity constraints, explicit equality, skolemization, reification, etc. – are orthogonal (i.e., independent/modular) relative to each other. And most of these generalizations/features are orthogonal to adding action and test expressions within rules cf. the earlier described Production LP KR extension. Moreover, one can anticipate that in future there will continue to be more KR generalizations/features developed for LP overall. (Likewise, there are interesting KR specializations/restrictions for LP, not just generalizations/features.)

We adopt a similar style, and thus say that "production logic programs" denotes some KR within that space of generalizations, such that action and test expressions are included. Hence, we will often speak of the "PLP approach" since it is not a single KR but rather a space of KR's.

4 Production Rules: New KR Semantics and Interoperability via Translation to PLP

In this section, we describe in medium detail a third new aspect of the Production LP approach: its formulation of a new declarative KR semantics for (a substantial subset

of the expressiveness of) production rules, via translation to the Production LP KR and thence to other rule systems that interoperate via the Production LP KR, notably Prolog and RDBMS systems, e.g., XSB. The translation enables interoperability and KB merging among all these systems.

This third aspect has been previously partially described in papers and presentations about SweetRules/SweetJess, as well as its open source code and documentation. These used the Situated extension of LP KR, rather than the Production extension of LP KR, and lacked all but a sketchy treatment of the theory guaranteeing the semantic equivalence of the translations.

There is an extensive literature on declarative logic programs including a number of expressive extensions. This provides well defined KR semantics (based on models) for Prologs and relational database management systems (RDBMS), and associated KR theory. This theory includes theory about semantics-preserving translation – and thus semantic interoperability – between different Prolog and RDBMS languages/systems, that captures a very substantial subset of the expressiveness of those languages/systems.

However, previous to our work on Production Logic Programs, including SweetJess [Grosof *et al.*, 2002] [Grosof *et al.*, 2005] (see especially the SweetJess component of SweetRules), there was not a similar style of declarative KR semantics formulated for production rules. Nor was there a theory of semantics-preserving translation between production rule languages/systems and Prolog or RDBMS languages/systems. Thus there was not theory available to support substantial semantic interoperability between one production rule language/system and another production rule language/system, nor between production rule languages/systems and Prolog or RDBMS languages/systems.

The same holds for event-condition-action (ECA) rules – i.e., when in the previous paragraph "production rule" is replaced everywhere by "event-condition-action rule". ECA rules are expressively fairly similar to production rules. They differ mainly by providing specialized expressiveness and control mechanisms for certain kinds of conditions, and for incremental update-driven behavior.

Particularly vital to a semantic formulation of production rules, yet previously lacking, are to give a semantic treatment (in the manner of declarative KR) of two aspects:

1. their procedural attachments for actions and tests; and

2. their basic rule-processing cycle, which is defined as a high-level procedure involving working memory, pattern matching, activation, agenda maintenance, firing, etc.

The PLP approach provides a declarative KR semantic formulation for a very substantial subset of production rules expressiveness, and the theory for semantic in-

teroperability between production rule languages/systems and Prolog/RDBMS languages/systems, as well as between different production rule languages/systems.

To achieve this, the PLP approach includes not only a newly generalized LP KR but also new associated theory about semantics-preserving translation between production rules and Prolog/RDBMS.

Next, we describe the semantics-preserving translation of production rules to and from the PLP KR. Of course, the best one can hope for is translation and interoperability with an expressively restricted *subset* of production rules.

Also, what *exactly* is "production rules", anyway? To our knowledge, there is no generally-accepted precise definition. As a first step towards defining translation, it is thus necessary to define a restricted production rule system whose details are abstracted away from the particulars of individual production rule languages/systems, i.e., CLIPS, Jess, etc. Note that we consider for now only the forward-chaining/forward-direction kind of rules in production rule systems; many production rule systems also have a kind of goal-driven backward-/mixed- direction kind of rules as well.

Accordingly, we define such an *abstract core production rule system (PR1)*, which includes a language, a knowledge base, and an engine. This is relatively straightforward, and a bit tedious, so in this section we omit most details and just cover some highlights. See our separate paper for full details of the definition [Grosof, 2005].

The rules are restricted to have in the head (i.e., RHS) only atoms that are either an `assert` of a fact or an action whose side effects are *external* to the KB and engine process. The agenda control strategy (ACS) is defined as a partial ordering over the set of potential activations, i.e., over the set of ground instances of the (KB's) production rules. The rules in PR1 are restricted to be definite, i.e., to have a non-empty head. As usual in current commercial production rule languages (e.g., Jess), PR1 rules are also restricted to be head-safe, negation-safe, sensor-safe, free of logical functions (sometimes a.k.a. the *Datalog restriction*); and sensor aproc's are always all-bound. *Head-safe* means that every logical variable appearing in the head/RHS of a rule also appears in the body/LHS of that rule. *Negation-safe* means that, within a rule, every logical variable appearing in a (body) negated atom (i.e., in a `not`'d atom) also appears in a non-negated atom in that rule's body.

PR1 production rules correspond straightforwardly to production rules in Jess (or CLIPS, which is quite similar), i.e., can be translated straightforwardly to and from Jess (or CLIPS).

In syntactic form, the PR1 rules correspond expressively to PLP rules in a straightforward manner. The key to the translation theory is to show that the PR1 engine process corresponds to the semantic construction of PLP, under suitable restrictions.

The theory consists of several major steps, starting from more restricted rules and proceeding to expressively generalize.

The first step of the theory treats the case of (definite, equality-free, function-free) Horn-form rules, without actions or tests or explicit equality. (Horn-form means there

is a single atom in the head, and a conjuction (`and`) of zero or more atoms in the body.) The insight here is that the engine process then iterates in a manner similar to (although differently sequenced from) the semantic construction of (definite, equality-free, function-free) Horn LP, resulting in a set of conclusions that is the same as the LP minimal model. The production rule system's behavior corresponds to doing exhaustive forward-direction inferencing in LP.

The second step of the theory adds actions and tests. The insight here is that PR1 then behaves similarly to the semantic construction of Production LP's treatment of actions and tests. Equality in the body is added also, handled in a manner similar to tests.

The third step of the theory adds negation. The insight here is that the behavior of negation (`not`) in production rules corresponds to the behavior of scoped default negation (a.k.a. well-founded negation-as-failure) in (Production) LP – but only under the restriction that agenda control strategy is ***stratified*** in the sense well studied in the theory of LP (particularly, in the well understood semantics of negation in LP).

Interestingly, the understanding of stratification of negation and its desirability was developed in the LP literature only in about 1985, several years after the basic "DNA" of production rule systems (including OPS5 and the Rete algorithm) was formed in the early 1980's. Our translation theory enables the "retrofitting" of production rules to incorporate these insights about the desirable semantics for negation – including intuitive behavior and algorithmic techniques – based on the understanding of stratification.

The fourth step of the theory adds enhanced logical connectives and quantifiers, in roughly the manner of Lloyd-Topor, including: `or`, `and`, and `exists` in the body; nesting of all these and `not` in the body; and `and` in the head. This is defined via transformations that reduce to the case without these, in a manner that is similar, but not identical, to the usual Lloyd-Topor (treatment of) logical connectives and quantifiers in LP. One main difference from Lloyd-Topor is in regard to the `or` connective when the rule has an action head; production rules behavior (e.g., in Jess) is defined as generating a set of "*subrules*", one per OR branch. This subrule generation may lead in worst case to an exponential blowup in the size/number of rules in the corresponding PLP, when multiple `or`'s nested within `and`'s. The other main difference from Lloyd-Topor is in regard to the `exists` connective, which is a kind of twice-nested `not`.

The above translation from (somewhat restricted) PR1 production rules to PLP is highly scaleable. The computational complexity of translation is tractable (i.e., worst-case polynomial-time), when the appearance of the `or` connective is restricted to prevent the potential blowup described above.

Production rules (situated) inferencing can even be done by translating to PLP, then inferencing in PLP, then translating the conclusions back from PLP to production rules. The next section describes translation from the corresponding case of PLP back to production rules (PR1); that translation also has tractable computational com-

plexity.

The computational complexity of inferencing in the corresponding case of PLP is tractable, given the VB and TA restrictions. VB means that there is a (constant) bound on the number of distinct logical variables appearing in a rule. TA means that the computational complexity of an aproc call is tractable. Both of these are quite reasonable, non-onerous restrictions. VB is typically met in practice. TA is usually met in practice. (One way to help ensure the TA restriction is to assume a (constant) bound on the size of a situated atom.)

There are several additional expressive features of production rules that appear relatively ripe for capturing via semantic translation to PLP.

Our current work on semantic translation of production rules to PLP includes extending the abstract production rule system to include retracting and updating of facts. Such retracting and updating of facts in production rule systems can arise from rule heads. It can also arise from the surrounding programming language environment (e.g., Java in the case of Jess) that provides facts and events that change those facts (notably, "dynamic shadow facts"). E.g., after a manager issues a stock-clearance notice, the price of various product items is marked down by twenty percent at the beginning of each week. This expressiveness for specification of incremental behavior is frequently used in applications of production rule systems and thus seems important to capture. Our approach employs the Courteous feature of LP to represent such retracting and updating in a modular and fairly natural fashion. The Courteous feature includes integrity constraints, e.g., to specify that there is only one value of the price per item. It also includes priorities, e.g., to specify that a rule or fact about the marked-down price takes precedence over a rule or fact about the old higher value of the price.

Another aspect remaining for future work is to treat what Jess calls "connective, predicate, and return-value constraint expressions". These appear to reduce essentially to `and`, `or`, `not`, and `test` expressions.

5 Translating Highly Expressive PLP to Production Rules

In this section, we describe briefly a fourth new aspect of the Production LP approach: the semantic translation of highly expressive PLP to production rules (PR1).

The case of PLP that corresponds closely syntactically to PR1 can be translated to PR1. (Note that, as mentioned earlier, PLP behaves differently than the syntactically corresponding PR1 for the case when there are `or` in rules with action heads. However, that case of PLP can still be translated fairly straightforwardly to PR1, and vice versa.)

Such translation, for a large subset of such PLP, has been implemented in SweetRules V2.1 as a translation to Jess from Situated LP (with similar essential expressiveness to the pertinent subset of Production LP). This uses, in part, a (tractable) translation/transformation that reduces sensor and effector statements, to sensor and

effector atoms within rules.

A crucial aspect of translating from PLP with negation to PR1 is to ensure that the agenda control strategy (ACS) in PR1 is stratified. An elegant way to specify such stratification in the ACS is to assign appropriate *salience* to the production rules. (Salience is an expressive feature of production rule systems.) A relatively simple algorithm suffices to analyze the predicate dependency graph (PDG) of the PLP rules, then determine a stratification partial ordering on the predicates and hence on the rules (and detect if the stratified restriction is violated). Doing this has tractable computational complexity. SweetRules V2.1 implements this technique.

Furthermore, a number of useful PLP expressive features for which there is no direct correspondent in production rules, can be translated into PR1 with KR semantics preserved.

Webized (URI) names of predicates and individual constants can be translated straightforwardly into (and then back out of) PR1. This has been implemented in SweetRules V2.1.

Webized aproc's are more complicated. In the case of Java/C++/C# aproc's, this asks for a standardized web-naming scheme for such methods; as far as we are aware, there are none quite yet. In the case of WSDL aproc's, this is easier; however, Jess and other production rule systems are still in process of developing more direct mechanisms to support WSDL. SweetRules V2.1 implements (webized) WSDL effectors, as well as a web-naming scheme for Java effectors and sensors, and supports these in its translation to Jess.

The Courteous feature represents prioritized defaults and conflict handling enforcing consistency with respect to mutual-exclusion integrity constraints (e.g., that something cannot be both a lion and a tiger, or that a given product cannot have two different prices). It also represents strong negation, a.k.a. "limited classical" negation, e.g., policy rules about both what is permitted and what is prohibited. The Courteous feature can be tractably expressively reduced, i.e., transformed, to just (default) negation, with tractable computational complexity. This has been implemented in SweetRules V2.1.

The rest of the Lloyd-Topor feature's expressiveness for enhanced logical connectives and quantifiers can also be tractably expressively reduced, i.e., transformed to just (default) negation, with tractable computational complexity.

The general, i.e., non-stratified, case of negation is significantly more complex to translate to production rules. SweetRules V2.1 implements a sophisticated, tractable technique that does this, again translating to Jess. However, it exploits some of the engine/control mechanisms of production rules that go beyond PR1 expressiveness, such as "modules" (a kind of multiple cooperating rulebases and engine threads). The technique is based on a novel bottom-up algorithm for computing the well-founded model of a LP, and has several optimizations for production rule systems. In current work, we are developing a new, second technique that after translation keeps within PR1 expressiveness.

The computational complexity of the above expressive PLP (situated) inferencing (including the Courteous and general negation features, in particular) is tractable, given the restriction to function-free, VB, and TA. As mentioned above, the computational complexity of the translation from such PLP to PR1 is tractable. And translation from PR1 back to PLP is tractable, as mentioned in the last section. Thus (situated) inferencing in such PLP can be performed via translating to production rules, then doing inferencing in production rules, then translating the conclusions back to PLP. (This is probably tractable, but requires a detailed tractability analysis of the overall production rules engine algorithms not just Rete, the literature about which we have not yet investigated.)

A number of additional expressive LP features, such as datatypes, reporting-style integrity constraints, Frame syntax, Hilog, reification, can also be tractably reduced, i.e., eliminated, by transformation to simpler LP, under suitable restrictions. These additional features can thus be translated tractably to production rules. These may be the basis for future further expressive expansion of the abstract production rule system expressiveness beyond PR1.

6 *Outline of Additional Sections Planned*

Next, we give a brief outline of additional sections we plan to add to this document.

- **Using Interoperability for Inferencing**: semantic interoperability provides a choice of inferencing engines, e.g., as SweetRules implements.
- **Support in PLP for Incremental Event-Driven Behavior**
- **Integrating with Ontologies**: several different ways to integrate ontologies with PLP are already known, including: ontological URI-references to predicates defined in background ontological knowledge bases, or import of ontological knowledge translated into PLP. This is where such ontological knowledge is defined in in rulebases, RDFS, OWL, FOL, or object-oriented/frame systems that employ default inheritance.
- **Inclusion Merging**: of knowledge bases, in PLP form (e.g., after translation into PLP from some other KR form)
- **Markup**: in XML, or in RDF
- **Implementation Techniques and Tools**: additional discussion, e.g., drawn from SweetRules and Flora-2
- **Application Scenarios**
- **Business Value Analysis**

- **Market Evolution Analysis**
- **Discussion and Future Work**

7 Conclusions

For now, see section 1.

8 Acknowledgements

The author wishes to thank Said Tabet, Harold Boley, Mike Dean, Michael Kifer, Shashidhara Ganjugunte, Hoi Y. Chan, Mahesh D. Gandhe, Ian Horrocks, Sumit Bhansali, Tim Finin, Peter Patel-Schneider, David Martin, Tim Berners-Lee, Abraham Bernstein, Stefan Decker, Raphael Volz, Boris Motik, Mark Musen, Martin O'Connor, Jos de Bruijn, Dieter Fensel, Tom Malone, Stuart Madnick, Sandro Hawke, James Bryce Clark, Patrick Gannon, Eric Prud'hommeaux, Rudi Studer, Eric Miller, Pat Hayes, and the participants in RuleML, SWSL, Joint Committee, WSML, and EECOMS efforts, as well as participants in the (2005) W3C Workshop on Rules for Interoperability, for collaborations and/or useful discussions on various aspects of the PLP approach. Support for recent phases of this work was provided by awards from DAML, the DARPA Agent Markup Language program. Thanks to DAML's very active program managers Mark Greaves, Murray Burke, and James Hendler for their encouragement and useful discussions. Support for earlier phases of this work was provided by IBM, an award from the NIST Advanced Technology Program, and awards from the Center for eBusiness @ MIT Vision Fund.

9 References

References

[Angele *et al.*, 2005] Jurgen Angele, Harold Boley, Jos de Bruijn, Dieter Fensel, Pascal Hitzler, Michael Kifer, Reto Krummenacher, Holger Lausen, Axel Polleres, and Rudi (alphabetically) Studer. Web Rules Language, Version 1.0. Technical report, Web Services Modeling Language (WSML) Working Group *et al*, Aug. 2005. Standards design specification and documentation. Available on the Web at: `http://www.wsmo.org/wsml/wrl/`. Also an acknowledged member submission to the World Wide Web Consortium (W3C).

[Battle *et al.*, 2005] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin,

Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said (alphabetically) Tabet. Semantic Web Services Framework, Version 1.0. Technical report, Semantic Web Services Initiative, May 2005. Standards design analysis, specifications, and documentation. Available on the Web at `http://www.daml.org/services/swsf/1.0/`. Also an acknowledged member submission to the World Wide Web Consortium (W3C).

[Bechhofer *et al.*, 2004] Sean Bechhofer, Mike Dean, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Guus Schreiber, and Lynn Andrea (alphabetically) Stein. OWL Web Ontology Language Reference. Technical report, World Wide Web Consortium (W3C), Feb. 2004. W3C standards Recommendation. Available on Web at: `http://www.w3.org/TR/owl-ref/`.

[Berners-Lee, 2005] Tim Berners-Lee. Primer: Getting into RDF & Semantic Web using N3, Aug. 2005. Working Paper. Version of Aug. 2005. Available on Web at: `http://www.w3.org/2000/10/swap/Primer/`.

[Bhansali and Grosof, 2005] Sumit Bhansali and Benjamin N. Grosof. Extending the SweetDeal Approach for E-Procurement using SweetRules and RuleML. In *Proc. First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, pages 113–129. Springer-Verlag, 2005. Lecture Notes in Computer Science LNCS 3791. Conference info available at: `http://2005.ruleml.org`. Held 10–12 November 2005, Galway, Ireland.

[Boag *et al.*, 2005] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jerome (alphabetically) Simeon. XQuery 1.0: An XML Query Language. Technical report, World Wide Web Consortium (W3C), Nov. 2005. W3C standards Candidate Recommendation. Available on Web at: `http://www.w3.org/TR/xquery/`.

[Boley *et al.*, 2005] Harold Boley, Benjamin N. Grosof, Said Tabet, and *et al* (alphabetically). RuleML: The Rule Markup Initiative, Dec. 2005. Standards design specifications and documentation. Version 0.9. Available on the Web at `http://www.ruleml.org`. Revised from Version 0.7 of Jan. 2001.

[Chen *et al.*, 1993] W. Chen, M. Kifer, and D.S. Warren. HiLog: A Foundation for Higher-Order Logic Programming. *Journal of Logic Programming*, 15(3):187–230, Feb. 1993.

[CIIMPLEX, 1999] CIIMPLEX. EECOMS: Extended Enterprise coalition for integrated COllaborative Manufacturing Systems, 1999. A $29 Million Advanced Technology Program project of the U.S. National Institute of Standards and Technology (NIST). By the CIIMPLEX industry consortium, 1998–2000. Info available on the Web at: `http://www.research.ibm.com/rules/eecoms.html`.

[Friedman-Hill and *et al*, 2004] Ernest Friedman-Hill and *et al*. Jess, 2004. Semi-open source software and documentation. Version 6.1p8. Available on Web at: `http://herzberg.ca.sandia.gov/jess/`.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. International Conference on Logic Programming (ICLP-88)*, pages 1070–1080, 1988.

[Grosof and Dean, 2005] Benjamin N. Grosof and Mike Dean. Semantic Web Rules with Ontologies, and their E-Service Applications, Nov. 2005. Detailed slideset, including bibliography, of 3.5-hour conference tutorial at the Fourth International Semantic Web Conference (ISWC-2005), held Galway, Ireland. Available on Web at: `http://ebusiness.mit.edu/bgrosof/#ISWC2005RulesTutorial`.

[Grosof and Poon, 2004] Benjamin N. Grosof and Terrence C. Poon. SweetDeal: Representing Agent Contracts with Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions. *International Journal of Electronic Commerce (IJEC)*, 8(4):61–98, Jul. 2004. special issue on web e-commerce.

[Grosof *et al.*, 1999a] Benjamin N. Grosof, Hoi Y. Chan, and `et al`. IBM CommonRules, Version 1.0, Jul. 1999. Software and documentation. Available on IBM AlphaWorks. Current version (V3.3+) and some past versions available on Web at: `http://www.alphaworks.ibm.com/tech/commonrules`.

[Grosof *et al.*, 1999b] Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In Michael P. Wellman, editor, *Proceedings of the 1st ACM Conference on Electronic Commerce (EC-99)*. ACM Press, 1999. Held in Denver, CO.

[Grosof *et al.*, 2002] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. SweetJess: Translating DamlRuleML to Jess. In *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002. (http://tmitwww.tm.tue.nl/staff/gwagner/RuleML-BR-SW.html) Held 14 June 2002, Sardinia (Italy) in conjunction with the First International Semantic Web Conference (ISWC-2002). Extended and updated Working Paper of May 2003 available at first author's website. Prototype available via http://www.daml.umbc.edu/sweetjess.

[Grosof *et al.*, 2003] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the 12th International Conference on the World Wide Web (WWW-2003)*. ACM Press, 2003. (http://www.www2003.org) Held May 20–23, 2003, Budapest, Hungary.

[Grosof *et al.*, 2005] Benjamin N. Grosof, Mike Dean, Shashidhara Ganjugunte, Said Tabet, Chitravanu Neogy, Dave Kolas, and *et al*. SweetRules: Tools for Semantic Web Rules and Ontologies, including Translation, Inferencing, Analysis, and Authoring, Apr. 2005. Open-source software, documentation, and samples. Version 2.1. Available on the SemWebCentral open source software repository, at `http://sweetrules.projects.semwebcentral.org`. Revised from Version 2.0 of Nov. 2004.

[Grosof, 2004] Benjamin N. Grosof. Representing E-Commerce Rules Via Situated Courteous Logic Programs in RuleML. *Electronic Commerce Research and Applications (ECRA)*, 3(1):2–20, Apr. 2004. special issue on semantic web and e-commerce.

[Grosof, 2005] Benjamin N. Grosof. Defining an Abstract Core Production Rule System, Dec. 2005. Working Paper. Version of Dec. 2005. Available on Web at: `http://ebusiness.mit.edu/bgrosof/#ACPRS`.

[Horrocks *et al.*, 2004] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rules Language Combining OWL and RuleML, Version 0.6. Technical report, Joint US/EU ad hoc Agent Markup Language Committee, Apr. 2004. Standards proposal research report. Available on the Web at `http://www.daml.org/2004/04/swrl/`. Committee was chaired by Mike Dean and co-led by Benjamin Grosof. Revised from Version 0.5 of Nov. 2003. Also an acknowledged member submission to the World Wide Web Consortium (W3C).

[Kifer and *et al*, 2005] Michael Kifer and *et al*. FLORA-2: An Object-Oriented Knowledge Base Language, May 2005. Open-source software, documentation, and samples. Version 0.94. Available on the SourceForge open source software repository, at `http://flora.sourceforge.net`.

[Kifer *et al.*, 1995] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, 42:741–843, 1995.

[Li *et al.*, 2003] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. *ACM Transactions on Information Systems Security (TISSEC)*, 6(1), Feb. 2003.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming, second edition*. Springer, Berlin, Germany, 1987.

[Prud'hommeaux and Seaborne, 2005] Eric Prud'hommeaux and Andy (alphabetically) Seaborne. SPARQL Query Language for RDF. Technical report, World Wide Web Consortium (W3C), Nov. 2005. W3C standards Working Draft. Available on Web at: `http://www.w3.org/TR/rdf-sparql-query/`.

[Przymusinski, 1994] Teodor Przymusinski. Well Founded and Stationary Semantics of Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.

[Van Gelder *et al.*, 1991] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of ACM*, 38(3):620–650, Jul. 1991.

[Warren, 2005] D.S. *et al* Warren. XSB, Mar. 2005. Open source software and documentation. Version 2.7.1. A Logic Programming and Deductive Database System. Available on Web at: `http://xsb.sourceforge.net`.

[Yang and Kifer, 2003] Guizhen Yang and Michael Kifer. Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. *Journal on Data Semantics*, pages 69–98, Sep. 2003. Lecture Notes in Computer Science 2800, Springer Verlag.