

# PSOA Prova: PSOA Translation of Pure Production Rules to the Prova Engine

Lukas Grätz<sup>1</sup>, Harold Boley<sup>2</sup>, and Adrian Paschke<sup>3</sup>

<sup>1</sup> Institut für Philosophie,  
Universität Leipzig, Germany  
lukas[DT]graetz[AT]studserv.uni-leipzig.de

<sup>2</sup> Faculty of Computer Science,  
University of New Brunswick, Fredericton, Canada  
harold[DT]boley[AT]unb.ca

<sup>3</sup> Fraunhofer FOKUS and Freie Universitaet Berlin, Berlin, Germany  
adrian[DT]paschke[AT]fokus.fraunhofer.de

**Abstract.** PSOA Prova enriches PSOA RuleML with parts of Reaction RuleML. It is implemented by combining PSOATransRun and Prova, a Prolog-based language and engine which supports object orientation, reactive programming as well as several other programming paradigms. A modified Prova engine targeted by PSOA RuleML's PSOATransRun translators is introduced and then extended to top-level assert and retract by allowing KB consult and unconsult at runtime. PSOA is further extended to pure production rules, a conclusion-asserting subset of Production RuleML, hence Reaction RuleML. This is exemplified for a PSOA Prova knowledge base about the British "Succession to the Crown Act 2013". These extensions yield a PSOA Prova language and engine available on GitHub. The PSOA Prova concepts are demonstrated with three formalization approaches for the British "Succession to the Crown Act 2013".

## 1 Introduction

In knowledge representation and reasoning, a Knowledge Base (KB) can be used to represent and reason about an unchanging world. Such an *immutable* KB, e.g. in a Pure Prolog-like rule language, can be implemented with a derivation-rule engine.

However, what if the state of the world changes? In this scenario, a *mutable* KB is needed, which can be successively modified to reflect each new state of the world. For a *static(ally)* (mutable) KB, these modifications are done (interactively) by the user with commands such as `consult`, `assert` and `retract`, before starting (each round of) querying. For a *dynamic(ally)* (mutable) KB [1,2], modifications can also be done (automatically) by reactive rules invoking such commands at runtime. In the following, we will only consider mutable KB facts, although Reaction RuleML also allows mutable KB rules.

The well-known relational databases correspond to mutable KBs of, essentially, (ground) facts, since these databases allow to insert and remove records,

usually in a static manner. Relational databases are broadly used as underlying data management systems on the web. For instance in eCommerce, the database for web shops must allow to interactively add and remove product records, when new products become available and others get sold out.

The PSOA RuleML language [3–6] has so far been presented with immutable KBs, corresponding to its reference implementation PSOATransRun 1.3.2-a<sup>4</sup>, but some of its use cases would call for a mutable KB, where new data and knowledge can be integrated into an existing KB. Port Clearance Rules [7] is an example: Whenever a ship seeks port clearance, a query is passed to PSOATransRun, which employs derivation rules to answer whether the ship is allowed to enter the port. If we want to reason about a new ship, a new KB fact representing the relevant information about the ship needs to be generated and then asserted into the Port Clearance KB. This is strictly monotonic and does not involve any production rules, only top-level `assert`.

In this paper, we slightly extend PSOA RuleML towards Reaction RuleML by a concept called *pure production rule* [8,9]<sup>5</sup>. This is, for example, motivated by the British “Succession to the Crown Act 2013” (Section 2). PSOA RuleML is used for representing the facts, rules and queries in the KB on an object-relational level of abstraction.

For demonstrating PSOA Prova, the PSOATransRun translator [10] is extended, in Section 3, to target Prova<sup>6</sup>, an ISO Prolog-augmenting scripting language written in Java. A further extension allows KB consult and unconsult at runtime. Three approaches for the succession law are demonstrated: The first version (Section 4) comes with a derivation rule only and is incomplete for the case that the successors’ parents dissolve their marriage. After extending PSOA to pure production rules, the second version replaces the derivation rule by the corresponding pure production rule in Section 5. Although unsound, version two provides a good insight into pure production rules. A sound and complete third version (Section 6) simulates an event condition action rule (ECA) on birth events with a pure production rule. Optimization approaches dealing with the evaluation time of pure production rules are discussed in Section 7. The paper concludes in Section 8. All PSOA Prova KBs can be found in Appendix A.

## 2 Use Case: Royal Family

Consider how the succession in a Royal Family works. In the United Kingdom, a successor to the British throne must be born in a legitimate marriage and be of Protestant faith. Here, “legitimate” means (for the first six successors) that the marriage has to be accepted by King or Queen. This regulation seems somehow complicated, but makes some sense: The future King or Queen should

<sup>4</sup> [http://wiki.ruleml.org/index.php/PSOA\\_RuleML#PSOATransRun](http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun)

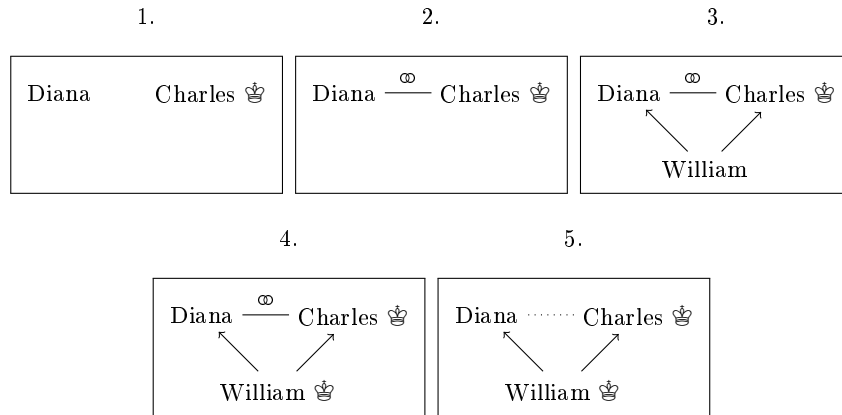
<sup>5</sup> <https://lists.w3.org/Archives/Public/public-rif-wg/2008Jun/0244.html>


<sup>6</sup> <https://github.com/ag-csw/prova>

be legitimated as a politician and it is necessary that – as the head of the Anglican Church – he or she is of Anglican confession.<sup>7</sup>

Our goal is to formalize the succession law and represent all successors in a Knowledge Base (KB). The KB will change in time, at least when successors marry or have children. You may call this time-dependent transition behavior a social-relationship flow. The following example describes how Prince William becomes a successor (also illustrated by Figure 1):

1. We start with a KB, given as a snapshot before Prince Charles and Lady Diana get married. This KB contains successors of the royal family as well as their marriage and child relationships which include the fact that Prince Charles is a successor.
2. Next, Diana and Charles get married and their marriage becomes legitimated. This is added to the KB.
3. Now, Diana gives birth to a child with Charles, Prince William.
4. The succession law is fulfilled, hence we can derive from the KB that William is a (legitimate) successor.
5. Finally, Charles and Diana dissolve their marriage. A divorce results in a modification of the relationships represented in the KB. This includes the removal (retraction) of the fact that Charles and Diana are in marriage.<sup>8</sup> However, Prince William remains a successor.



**Fig. 1.** Prince William becomes a successor (indicated by “”) after Charles and Diana are (legitimately) married (indicated by “ $\infty$ ”). The arrows indicate that William is a (biological) child of Charles and Diana.

<sup>7</sup> The current succession law is available here:

<http://www.legislation.gov.uk/ukpga/2013/20>

<sup>8</sup> If the KB should include divorces, then the fact that Charles and Diana are now divorced must be added, too.

Family trees are one of the most prominent introductory examples for knowledge representation and reasoning in web ontologies as well as in rule-based systems: In description logics, family trees illustrate the basic language *ALC*, but family trees also have a long tradition in rule languages like Prolog, as they are used in several introductory textbooks; even the succession to the British throne has advantages for students learning Prolog [11]. PSOA RuleML was also motivated by family examples [3, 4, 12].

In fact, genealogy databases are an important application for knowledge representation in the web. The family history of some famous artists, musicians and scientists or the history of aristocratic families are use cases for genealogy knowledge. Although these family trees can include thousands of individuals, historians and the public are highly interested in getting the family relationships in a nice format.

And if you have such a large KB, you will want to get information beyond the traditional family relationships common to all family trees (married, parent, child, male, female): The succession to a throne hierarchy is an example that is only relevant for royal families. In the current paper we demonstrate a fraction of a proposed British Royal Family KB by representing selected facts about Diana, Charles and William. This way we keep the focus on the development of derivation and production rules.

### 3 PSOA Prova Translator Implementation

This section deals with the underlying technical part of the PSOA Prova project.

The PSOA Prova translator<sup>9</sup> is based on the ANTLR-based PSOATransRun translator. PSOATransRun translates PSOA KBs and queries (given in PSOA Presentation Syntax) either into TPTP or XSB Prolog by doing several normalization steps (unnesting, subclass rewriting, objectification, description and skolemization, conjunctive-conclusion splitting, flattening) [5].

Although Prova does not strictly conform to ISO Prolog, it is in fact a Prolog implementation. Prova's goal is to be a powerful language by supporting many programming paradigms simultaneously, so the ISO Prolog standard is of minor importance. Nevertheless, both the normalization steps and the Prolog translator are reused for the Prova Translator. Some basic test cases passed with only a few syntactic modifications in the output of the original translator.

The Java-Prolog communication is different from the XSB and SWI Prolog targets. The latter two require the middleware Interprolog. Prova has a Java API and is implemented in Java, hence this target no longer needs to depend on Interprolog. PSOA Prova's communication inside of PSOATransRun is done in the ProvaEngine class, part of the `org.ruleml.psoa.psoatransrun.prova` package. The ProvaEngine extends ReusableKBEngine, which means that both KB and ProvaEngine can be reused for several queries. When the method `loadKB` with the translated KB as argument is called from outside, a new Prova instance

<sup>9</sup> [http://wiki.ruleml.org/index.php/PSOA\\_Prova](http://wiki.ruleml.org/index.php/PSOA_Prova)

is created as a `ProvaCommunicatorImpl` object. There is no need to store the translated KB separately in a file, because a simple copy of the KB string is sufficient for the Prova instance.

Query strings are passed to the `executeQuery` method. They typically consist of a conjunction of goals with free variables. Each answer set consists of an assignment of free variables. In Prova, goals are solved by the built-in predicate `'solve/1'`. Only one goal is supported at once: This is why a query consisting of a conjunction of multiple goals like

```
a(...), b(...), c(...)
```

is first transformed to a rule

```
query(...) :- a(...), b(...), c(...).
```

and then the single goal `'query(...)` is solved by the built-in `'solve/1'`:

```
:- solve(query(...)).
```

### 3.1 Runtime KB Consult and Unconsult

If you call `PSOATransRun` from the command line,

```
$ java -jar PSOATransRunLocal.jar -i RoyalFamily-KB1.psoa -s -l prova -p
Translated KB:
menterm('_Charles', '_successor').
>
```

`PSOATransRun` translates and executes the PSOA KB file given as command-line argument. The `“-p”` echos the translated Prova KB (the original KB could be echoed by `“-e”`). Now you can enter your queries, which will be answered by `PSOATransRun` and the underlying rule engine (for reasonable queries see Section 4).

PSOA Prova extends `PSOATransRun` by supporting loading and unloading PSOA KBs at runtime, but please note that this feature is currently highly experimental. In PSOA Prova,

```
> consult RoyalFamily-KB2.psoa
```

is not interpreted as a query but as an import statement:

```
Translated KB:
prdsloterm('_1', '_marriage', '_partner', '_Diana').
prdsloterm('_1', '_marriage', '_partner', '_Charles').
menterm('_1', '_marriage').
```

Queries will then be interpreted on the basis of a unified KB that includes the KB given in command line and every KB that was consulted in query mode. It is possible to consult multiple KBs and ask queries in between. At any time, you could remove a KB that was consulted in query mode by running:

```
> unconsult RoyalFamily-KB2.psoa
```

Now you can check that the KB was removed successfully (observe that, because of the translation into a conjunction, the order of the partners does not matter):

```
> marriage( partner+>Diana partner+>Charles )

Translated Query:
prdsloterm(Q1,'_marriage','_partner','_Diana'),
prdsloterm(Q1,'_marriage','_partner','_Charles'),
memterm(Q1,'_marriage').

Answer(s):
No
```

Consulting and unconsulting KBs was made possible by Prova API functions

```
consultSync( KB, key, ... );
```

for consulting a KB and

```
unconsultSync( key );
```

for removing the consulted KB referenced by “key”. In the PSOA Prova implementation, the key corresponds to the file path of the original PSOA KB.

Note that removing a KB is not necessarily equivalent to a rollback of the transaction of consulting, if the language allows some reactive rules. For example, if we have a rule that results in printing a physical paper sheet, this cannot be undone. Pure production rules as defined in Section 5 exploit this by persistently asserting some derived facts.

## 4 Succession by a Derivation Rule

This section reconstructs the time-dependent mutable KB of Section 2 in PSOA, step by step. At this point, we have to specify the succession law. If the parents are in marriage the following simple rule derives the ‘successor’-membership from a child of a successor (again, the order of the ‘parent’ slots is irrelevant).

```
Forall ?Ch ?P1 ?P2 (
  ?Ch#successor :- And( ?Ch#child( parent->?P1 parent->?P2 )
                       marriage( partner+>?P1 partner+>?P2 )
                       ?P1#successor )
)
```

1. We start with a KB that contains successors of the royal family as well as their marriage and child relationships. In reality, we would start with a snapshot of a huge KB, but for the demo the `derivation-KB-snapshot.psoa` only contains the succession derivation rule and the successorship of Charles. Try it by running:

```
$ java -jar PSOATransRunLocal.jar -s -l prova -i derivation-KB-snapshot.psoa
```

You can test that Charles is a successor by entering:

```
> Charles#successor
```

This tests whether the OID `'Charles'` is member of the predicate `'successor'`. The actual hierarchy, which means the number in the line of succession could be indicated by a dependent slot `'number+>'`:

```
> Charles#successor( number+>1 )
```

The answer is `'No'`, because this is not implemented. Moreover, the succession number could change, for example, if an elder brother's child would get inserted in the succession line before the younger brother. Therefore, it seems more natural to implement the succession line as a dynamic list using the dependent slots `'previous+>'` and `'next+>'`. For simplicity, the demo deals with the membership of the succession line only. It is still an interesting question who is part of the succession line in a Royal Family.

2. Next, Diana and Charles get married and their marriage becomes legitimated. This is represented in the KB by adding

```
> marriage( partner+>Charles partner+>Diana )
```

which is exactly the content of `'RoyalFamily-KB2.psoa'`:

```
> consult RoyalFamily-KB2.psoa
```

3. Now, Diana gives birth to a child with Charles, Prince William:

```
> William#child( parent->Charles parent->Diana )
```

We get this after:

```
> consult RoyalFamily-KB3.psoa
```

Both parents are given by the independent slot `'parent->'`.<sup>10</sup>

4. The succession law is fulfilled. Hence we can derive from the KB that William is a (legitimate) successor:

```
> William#successor
```

You could also want to test more advanced (e.g., non-ground) queries like

```
> ?Q#successor
```

or

<sup>10</sup> This is equivalent to `'William#child'`, `'William#Top( parent->Charles )'` and `'William#Top( parent->Diana )'` which we would get after some PSOA normalization (cf. [7]). It is conceivable that one parent is unknown. In this case, PSOA allows to formalize just the parentship to the second parent.

```
> And( ?Ch#successor ?Ch#child( parent->Diana ) )
```

The first query returns all successors (like Charles and William), the second query gives successors who are children of Diana.

5. Finally, Charles and Diana dissolve their marriage which can be represented by retracting

```
> marriage( partner+>Charles partner+>Diana )
```

from the KB. This is done by:

```
> unconsult RoyalFamily-KB2.psoa
```

Note that this was a condition of the successor derivation rule above, hence we can no longer derive:

```
> William#successor
```

However, Prince William should remain a successor. This is not guaranteed by the derivation rule, since the conclusion no longer holds.

So the question is: What went wrong with the reconstruction? The answer is that it is sufficient and necessary that the parents of a successor are in marriage at the time of birth and it is irrelevant what happens later. This could be implemented by a static immutable approach, where marriages are defined by a time period, so that even dissolved marriages remain in the KB. But, as we will see later, the time-dynamic formalization approach has some advantages over the static one. So, in the next section, let us fix the mutable approach.

## 5 Succession by a Pure Production Rule

As we saw in the previous section, the derivation rule for succession was suitable only if it was evaluated directly after birth. When the parents are divorced, the condition of this rule no longer holds. In a production rule system like CLIPS with an inflationary model, this would not be a problem, since inferred facts are stored persistently, even if conditions are eventually no longer satisfied. This gives us the idea to extend derivation rule languages to some sort of production rules.

So let us define an extended derivation rule where the inferred conclusion remains persistent. This rule is called pure production rule, as a general production rule “*if condition, do action*” usually allows more actions like retraction of inferred facts (besides the difference in the backward-reasoning semantics of logical derivation rules and the forward-directed classical production rules with inflationary semantics).

**Definition 1.** *A pure production rule is an extended derivation rule, where the derived conclusion is asserted persistently to the KB. If the condition holds, the conclusion is derivable; moreover, the conclusion will be asserted at least before the condition becomes unsatisfied.*

*The syntax is slightly modified from the derivation rule syntax by replacing ‘:-’ with ‘: :-’:*



```
<conclusion> :- <condition>
```

If a non-reactive rule-language is extended only to pure production rules, they have the same semantics as the standard implication of a derivation rule: We can still derive the conclusion because any asserted fact was derived at some point. But in case that the underlying KB changes (by top-level assert and retract), a pure production rule comes in handy: Even if the conditions are no longer satisfied the asserted fact stays in the KB. Note, that pure production rules only assert proven conclusions as facts. In case of unrestricted knowledge updates during the derivation proofs, we would need to handle side effects between assertions and retractions and roll-back resp. disallow any updates in failing proofs. This can be specified in (Reaction) RuleML semantic profiles with the @safety attribute set to “transactional” and implemented, e.g. with transaction logics.

The pure production succession rule is the same as the derivation rule of the previous section, except that “:-” is replaced by “::-”:

```
forall ?Ch ?P1 ?P2 (
  ?Ch#successor ::- And( ?Ch#child( parent->?P1 parent->?P2 )
                        marriage( partner+>?P1 partner+>?P2 )
                        ?P1#successor )
)
```

It is straightforward to extend PSOATransRun to pure production rules: Basically, the ANTLR-based tree walkers are adjusted to treat pure production implications ‘::-’ exactly as standard derivation implications ‘:-’. Only in the last PSOA to Prolog conversion step, assert of the conclusion of a pure production rule is appended to the condition list.<sup>11</sup> So

```
<conclusion> ::- <condition_1>, ..., <condition_n>.
```

is translated to:

```
<conclusion> :- <condition_1>, ..., <condition_n>, assert(<conclusion>).
```

Let us now retry the reconstruction of the relationship flaw illustrated by Figure 1 in Section 2 by using the pure production rule in contrast to the derivation rule of Section 4. First of all, we start with the modified KB snapshot of

<sup>11</sup> Implementation details (briefly):

1. New token `PRODUCTION` : ‘::-’; in `PSOAPS.g`
2. For Every Tree walker in `PSOACore` the ANTLR-rules are adjusted to behave the same for (standard) `IMPLICATION` and `PRODUCTION`.
3. For every pure production rule, `PrologConverter.g` in `PSOA2X` finally appends `asserta(<string-copy of the rule head>)`.

the current world of the year 1981, before the date of Charles's and Diana's marriage.

```
$ java -jar PSOATransRunLocal.jar -s -l prova -i production-KB-snapshot.psoa
```

We can test that the knowledge states 1. to 4. from Section 4 can be reproduced by the pure production rule:

1. ... 3. (see Section 4)
4. The succession law is fulfilled. Hence we can derive from the KB that William is a (legitimate) successor:

```
> William#successor
Yes
> ?Q#successor
?Q=_William ;
?Q=_Charles
> And( ?Ch#successor ?Ch#child( parent->Diana ) )
?Ch=_William
```

These queries evaluate the pure production rule 'successor': The conclusion is returned like in a normal derivation rule but additionally asserted.

5. Finally, Charles and Diana dissolve their marriage which can be represented by retracting

```
> marriage( partner+>Charles partner+>Diana )
Yes
```

from the KB. This is done by:

```
> unconsult RoyalFamily-KB2.psoa
```

Note that we retracted a condition of the successor production rule above. Nevertheless we can derive

```
> William#successor
Yes
```

because this was asserted as a new fact. The answer is correct according to the law of succession, since Prince William should remain a successor, although this is not guaranteed by the derivation rule anymore, since the conclusion no longer holds:

```
> marriage( partner+>Charles partner+>Diana )
No
```

## 6 Become Successor at Birth and not Afterwards!

If we switch 2. and 3. in the sequence displayed by Figure 1, the child still becomes a successor in the next step. This is problematic: For example, if Charles had a child with his second wife Camilla before he married Camilla, this child

would become successor too (according to our formalization). This was not intended, because the succession law clearly requires that the parents have to be married during birth.

So we have to distinguish between a legitimate child, whose parents are married during birth and an illegitimate child whose parents marry afterwards. Complex event processing (CEP), with temporal reactive semantics, would fix this problem, but we will see that this can be modeled by pure productions rules, too:

First, we change the condition of the derivation rule to use a length-1 tuple indicating legitimacy and to keep only the ‘successor’-typed ‘parent’ slot:

```
Forall ?Ch ?P1 (
  ?Ch#successor :- And( ?Ch#child( legitimate parent->?P1 )
                      ?P1#successor )
)
```

Then we have to define a pure production rule for legitimacy that will be evaluated only for newly born children:

```
Forall ?Ch ?P1 ?P2 (
  ?Ch#child( legitimate ) :-
    And( ?Ch#newly_born( parent->?P1 parent->?P2 )
         marriage( partner+>?P1 partner+>?P2 )
    )
)
```

These rules are included in `legitimate-KB-snapshot.psoa`:

```
$ java -jar PSOATransRunLocal.jar -s -l prova -i legitimate-KB-snapshot.psoa
```

This time, we have to adjust step 3 (for the steps 1-2 see Section 4):

### 3. Check that William is not yet born:

```
> William#child
No
```

Assert that William is born as child of Diana and Charles:

```
> consult RoyalFamily-KB3.psoa
```

Now explicitly assert that William is newly born:

```
> consult RoyalFamily-KB3-a.psoa
```

You can test this, if you like:

```
> William#newly_born
```

Be sure to evaluate the production rule to obtain that William is a legitimate child:

```
> William#child( legitimate )
```

Now, retract that William is newly born:

```
> unconsult RoyalFamily-KB3-a.psoa
```

4. The conditions for the successor rule are satisfied, hence we can derive

```
> William#successor
```

5. ... even if we retract the marriage:

```
> unconsult RoyalFamily-KB2.psoa
```

Note that we have presupposed in step 3 that consult and unconsult instructions are executed atomically: No other top-level consult and unconsult statements are allowed when the child is marked as newly born. It would be fatal if the parents marry while the child is still considered as newly born and we derive that the child is legitimate. On the other hand, queries are allowed at any time.

The atomicity can be implemented by mutual exclusion in a layer between PSOA Prova on the technical layer and the event processing layer.

Also note that this is kind of a simulation of an *ECA rule* (Event Condition Action): It is an event that a child is newly born, triggered for a short time interval. On this event, the production rule should fire that a child is legitimate, if the parents are married.

To make the formalization complete you could add some rules to derive that someone is a disqualified successor. These rules correspond to [11]. For example, a person with catholic faith is disqualified<sup>12</sup> from the succession line:

```
forall ?S (
  ?S#successor( disqualified ) :- Or( ?S#deceased
                                     ?S#successor( abdicated )
                                     ?S#catholic )
)
```

Finally, it is simple to get the actual succession number of each person with the depth-first search algorithm.

## 7 When to Evaluate Pure Production Rules?

If Charles and Diana dissolve their marriage, then this can be modeled by retracting the fact:

```
marriage( partner+>Charles partner+>Diana )
```

<sup>12</sup> It would be consequent in a way if deceased people were deleted from the KB (just like marriages). However, since the succession is stored (persistently) in the KB by a pure production rule, the deletion cannot be done by simply unconsulting a source file. Such a functionality is future research.

According to the evaluation of the pure production rule, Prince William remained a successor, although the conditions for the production rule were no longer satisfied. But if the Production Rule had never been evaluated before the marriage of Charles and Diana was dissolved, it can no longer be derived that William is a successor. This could be intended, but in our opinion, this would not result in a clear logical semantics. Whether a rule was evaluated or not is hardly predictable and should be invisible to the user.

A clearer and transparent functionality for retracting a fact could be achieved by evaluating all production rules depending on this fact right before the fact is retracted. This would lead to a non-monotonic state transition semantics, where the KB transits from one knowledge state to the next knowledge state. In the succession example, the fact that Prince William is a successor would be asserted when the marriage of Charles and Diana is retracted (or dissolved).

At the same time, a pure production rule is a special ECA Rule “*On Event, if  $g(\dots)$  then assert( $g(\dots)$ )*”. However, it is questionable what kind of event triggers this rule (cf. [13]), e.g.:

1. **Structure operation**
  - (a) after `assert`
  - (b) before `retract`
  - (c) ...
2. **Behavior invocation**
  - (a) when the head of a pure production rule is a derivation (sub)goal
  - (b) ...
3. **Clock** (e.g. polling)
4. **External** (e.g. by the CEP engine which orders `assert` and `retract`)
5. ...

Currently, only 2a is implemented for “`:-`”. The other possibilities are also feasible, when a PSOA pure production rule

```
<conclusion> :- <condition>
```

is translated not in the way of Section 5 but to

```
<conclusion> :- <condition>.
<event> :- <condition>, assert(<conclusion>).
```

in Prolog, where `<event>` may depend on `<condition>` and/or `<conclusion>`. Hence, it follows that any pure production rule is in fact an abbreviation for two rules, first, a Prolog derivation rule and second, an ECA rule that is evaluated on event (not visible to the knowledge engineer).

In classical production rule systems the update actions, such as `assert` and `retract`, can be considered as implicit events that act as trigger to the production rules. In the Prova implementation this reactive behaviour can be implemented using the Prova message built-ins that inform about updates to the knowledge base and act as event trigger. In our example, the retraction of the marriage triggers the additional reaction rule, which first asserts that William is a successor and then actually retracts the marriage (this would be implemented by 1b):

```
rcvMsg(XID,self,From, inform, retract(marriage("Charles","Diana")) :-
    marriage("Charles","Diana"),
    assert(successor("William")),
    retract(marriage("Charles","Diana")).
```

## 8 Conclusion and Outlook

PSOATransRun has been extended for PSOA Prova to allow consult and unconsult of KB modules at runtime. This is a first step to support (statically) mutable KBs.

In addition, PSOATransRun was extended to allow the derivation of persistent facts by so-called pure production rules. Pure production rules can be seen either as extended derivation rules (asserting conclusions) or, dually, as restricted production rules (with only assert actions).<sup>13</sup> They constitute the derivation-production-unifying core of rule languages such as RuleML: a minimal superset of Horn logic and production rules. Negations like negation-as-failure are not necessarily part of the core (and are not used in this document).

Both extensions were motivated and used by a time-dynamic knowledge transition for the use case of a royal family KB.

A common query for royal family KBs is to get all legitimate successors. This was discussed for famous members of the British Royal Family according to the ‘British Succession to the Crown Act 2013’. Three solution approaches were given in the paper: The first uses a classical derivation rule. The second a pure production rule. The third approach is more advanced and simulates an ECA rule by top-level assert, retract and a pure production rule.

All three approaches were tested against the scenario in how Prince William became a successor, because one of his parents is a successor and both parents were married at the time he was born. The first derivation-rule approach failed (hence, is incomplete) because it was no longer derivable that William was a successor after the parents were divorced.

Pure production rules were defined as extended derivation rules where the conclusion is also asserted persistently. For the second approach we could derive that William was still a successor after the parents were divorced, since this was asserted persistently by the pure production succession rule. As discussed in Section 6, this pure production formalization approach is unsound, since even someone who was not born as a successor could eventually become a successor: This is the case when the parents were not married during birth but married afterwards.

The third approach is based on the first approach and fixes incompleteness and unsoundness by defining a pure production rule for legitimate children.

Nevertheless, all approaches can be adequate for other royal families (with other rules) or in case that the succession law changes. In general, dynamic

<sup>13</sup> Also similar to rules for goal-directed bottom-up execution (for exhaustive bottom-up execution see, e.g., (OO) jDREW Bottom-Up: <http://www.jdrew.org/ojdraw/>).

KBs for time-dynamic domains have the advantage of not requiring an explicit representation of time-related aspects: It is not necessary to store any time or date references or historic revisions of the KB, even if the knowledge state, and therefore the KB, changes in time.

Prova, with its built-ins for sending and receiving messages supports implicit events that can trigger an evaluation of (sub)goals so that a forward-directed production semantics for knowledge updates can be implemented. To avoid non-declarative behavior, as discussed in Section 5 about inflationary semantics of classical production rules, this would additionally require that the logical semantics is extended to a transaction logic [14], where knowledge updates, such as assert and retract, are rolled back in case of failing proofs.

## References

1. Katerinenko, R.S., Bessmertnyi, I.A.: A method for acceleration of logical inference in the production knowledge model. *Programming and Computer Software* **37**(4) (2011) 197–199
2. Pujara, J., Getoor, L.: Building dynamic knowledge graphs. In: *NIPS Workshop on Automated Knowledge Base Construction*. (2014)
3. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: *Proc. 5th International Symposium on Rules: Research Based and Industry Focused (RuleML-2011 Europe)*, Barcelona, Spain. *Lecture Notes in Computer Science*, Springer (July 2011) 194–211
4. Boley, H.: PSOA RuleML: Integrated Object-Relational Data and Rules. In Faber, W., Paschke, A., eds.: *Reasoning Web. Web Logic Rules (RuleML 2015) - 11th Int'l Summer School 2015*, Berlin, Germany, July 31- August 4, 2015, *Tutorial Lectures. Volume 9203 of LNCS.*, Springer (2015)
5. Boley, H., Zou, G.: *Perspectival Knowledge in PSOA RuleML: Representation, Model Theory, and Translation*. *CoRR* **abs/1712.02869** (2017)
6. Zou, G.: *Translators for Interoperating and Porting Object-Relational Knowledge*. PhD thesis, Faculty of Computer Science, University of New Brunswick (April 2018)
7. Zou, G., Boley, H., Wood, D., Lea, K.: Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic. In: *Proceedings of the RuleML+RR 2017 Challenge*. Volume 1875., *CEUR* (July 2017)
8. Paschke, A., Boley, H.: Rules capturing events and reactivity. In Giurca, A., Gašević, D., Taveter, K., eds.: *Handbook of Research on Emerging Rule-Based Languages and Technologies*. Hershey, PA (2009)
9. Paschke, A., Boley, H., Zhao, Z., Teymourian, K., Athan, T.: Reaction ruleml 1.0: Standardized semantic reaction rules. [15] 100–119
10. Zou, G., Boley, H.: PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D., eds.: *Rule Technologies: Foundations, Tools, and Applications, Proc. 9th International Symposium, RuleML 2015*, Berlin, Germany, August 2-5, 2015. Volume 9202 of *Lecture Notes in Computer Science.*, Springer (August 2015)
11. Mohr, J.: Two novel prolog assignments. In Brézillon, P., Russell, I., Labat, J., eds.: *Proceedings of the 14th Annual SIGCSE Conference on Innovation and Technology*

- in Computer Science Education, ITiCSE 2009, Paris, France, July 6-9, 2009, ACM (2009) 350
12. Manir, M.S.A., Riazanov, A., Boley, H., Baker, C.J.O.: PSOA ruleml API: A tool for processing abstract and concrete syntaxes. [15] 280–288
  13. Paton, N.W., Díaz, O.: Active database systems. *ACM Comput. Surv.* **31**(1) (March 1999) 63–103
  14. Bonner, A.J., Kifer, M.: An overview of transaction logic. *Theoretical Computer Science* **133**(2) (1994) 205 – 265
  15. Bikakis, A., Giurca, A., eds.: Rules on the Web: Research and Applications - 6th International Symposium, RuleML 2012, Montpellier, France, August 27-29, 2012. Proceedings. Volume 7438 of Lecture Notes in Computer Science., Springer (2012)

## A Appendix

————— RoyalFamily-KB1.psoa —————

```
RuleML (
  Assert (
    Charles#successor
  )
)
```

————— RoyalFamily-KB2.psoa —————

```
RuleML (
  Assert (
    marriage( partner+>Diana partner+>Charles )
  )
)
```

————— RoyalFamily-KB3.psoa —————

```
RuleML (
  Assert (
    William#child( parent->Diana parent->Charles )
  )
)
```

————— derivation-KB-snapshot.psoa —————

```
RuleML (
  Assert (
    Charles#successor

    % Derivation rule for the succession to the British throne:
    % A successor must be a (biological) child of a successor and both
    % parents have to be in marriage.
    Forall ?Ch ?P1 ?P2 (
      ?Ch#successor :-
        And( ?Ch#child( parent->?P1 parent->?P2 )
            marriage( partner+>?P1 partner+>?P2 )
            ?P1#successor
          )
    )
  )
)
```



```

production-KB-snapshot.psoa
RuleML (
  Assert (
    Charles#successor

    % Pure production rule for the succession to the British throne:
    % A successor must be a (biological) child of a successor and both
    % parents have to be in marriage.
    Forall ?Ch ?P1 ?P2 (
      ?Ch#successor :-
        And( ?Ch#child( parent->?P1 parent->?P2 )
            marriage( partner+>?P1 partner+>?P2 )
            ?P1#successor
          )
        )
    )
  )
)

```

```

legitimate-KB-snapshot.psoa
RuleML (
  Assert (
    Charles#successor

    % Derivation rule for the succession to the British throne:
    % A successor must be a legitimate (biological) child of a successor.
    Forall ?Ch ?P1 (
      ?Ch#successor :-
        And(
          ?Ch#child( legitimate parent->?P1 )
          ?P1#successor
        )
      )
    )

    % Pure production rule for a legitimate child
    Forall ?Ch ?P1 ?P2 (
      ?Ch#child( legitimate ) :-
        And( ?Ch#newly_born( parent->?P1 parent->?P2 )
            marriage( partner+>?P1 partner+>?P2 )
          )
      )
    )
  )
)

```

```

RoyalFamily-KB3-a.psoa
RuleML (
  Assert (
    William#newly_born
  )
)

```