Simulating Production Rules Using ACTHEX*

Thomas Eiter, Cristina Feier, and Michael Fink

Institute of Information Systems, Vienna University of Technology,
Favoritenstraße 9-11, A-1040 Vienna, Austria
lastname@kr.tuwien.ac.at

Abstract. Production rules are a premier formalism to describe actions which, given that certain conditions are met, change the state of a factual knowledge base and/or effect a change of the external environment in which they are situated, based on an operational semantics. ACTHEX is a recent formalism extending HEX programs, such that the specification of declarative knowledge in the form of logic programming rules can be interleaved with a type of condition-action rules which prescribe the execution of (sequences of) actions that can change the external environment. Under the provision of a specific semantics of conditions, the operational semantics of production rules can be simulated using the model-based semantics of ACTHEX. Given that the latter features abstract access to external sources of computation, it can capture a range of concrete execution semantics and, moreover, facilitate access to heterogeneous information sources.

Keywords: ACTHEX, production rules

1 Introduction

Production rules are one of the most successful rule paradigms with regard to practical applications. Since more than three decades, they have been deployed to various domains (see e.g. [28] for an overview of early production systems); tools and platforms like JRules, Clips, Jess, or Drools have become popular. They are largely based on theoretical foundations and key technology that has been developed in the rise of the paradigm. In the recent years, there has been a reviving interest in foundational research on production rules, triggered by extensions like the combination of a production rule system (PRS) with ontologies or linked data, which surfaces in the context of the Semantic Web. Formalisms of W3C's Rule Interchange Format (RIF) working group address this, aiming to provide a minimalistic standard for combining rules and ontologies in the web ontology language (OWL) resp. RDF data.

Achieving such combinations is somewhat more difficult than similar combinations of logic-based rules and ontologies (see e.g. [21, 9] for surveys). The reason is that production rules have an operational semantics, in which a working memory (roughly, a factual knowledge base) is changed by the firing of rules which are matched against the current state of the working memory; out of the set of fireable rules, one is selected for execution according to some *conflict resolution strategy*. The "result" of the evaluation

* This work is partially supported by the Austrian Science Fund (FWF) under the project P20840, and by the European Commission under the project OntoRule (IST2009231875).

of a PRS is the final state achieved after repeating matching and execution, known as *recognize-act cycle*, as long as possible, provided that it terminates.

Elements of this procedure like a concrete conflict resolution strategy (possibly based on the execution history), and a change mechanism for removing information from the working memory, are challenging for a declarative formalization in standard logic-based formalisms, which lack procedural elements and are subject to logical ramifications: if a fact A is to be removed, presence of a fact B and a clause $B \rightarrow A$ in the background still allows to derive A, so mere physical removal of A is insufficient. Another issue is negation in rule conditions, which refers to the current state.

Several works have considered simulation of PRS in logic-based formalisms, facing these difficulties. E.g., Baral and Lobo [2] used logic programming under stable semantics and situation calculus notation; Kowalski and Sadri [19] resorted in an agent-flavored approach to abductive logic programming combined with reactive rules, on a database with destructive assignments. de Bruijn and Rezk [5] instead employed the mu-calculus and fixpoint logic. Damasio et al. [8] used incremental Answer Set Programming to realize the default semantics of the RIF-PRD dialect. Most recently, aiming at a combination of production rules and ontologies, Rezk and Kifer [24] use transaction logic, while Rosati and Franconi [27] describe a general framework for combining production rules and ontologies, with the purpose of studying termination and complexity for different classes of rules and ontologies.

These proposals often commit to particular settings and ramifying assumptions (e.g., a particular conflict resolution strategy and/or removal strategy), and actual execution of a simulated PRS run to obtain full emulation requires an extra effort. The support of access to other, heterogeneous information sources, or interaction with an environment in which the knowledge base is situated, usually requires an extension.

In this paper, we consider simulation of PRS in the ACTHEX formalism, in which actual execution and the above extensions are readily achieved. ACTHEX [3] is a logic-programming based framework for declarative specification of action execution, using a rule-based language that builds on special action atoms. It is a relative of Gelfond and Lifschitz's action languages [17], in which actions and their effects are typically described in rule-based languages; however, in ACTHEX the effect of actions, especially on the environment, might be implicit via the plugins used. Briefly, ACTHEX allows a) to express and infer a predictable execution order for action atoms; b) to express soft (and hard) preferences among a set of possible action atoms; and c) to actually execute a set of action atoms according to a predictable schedule. ACTHEX extends HEX-programs [11], which in turn extend non-monotonic logic programs under answer set semantics with external atoms. The latter model access to external computation via an abstract interface and a plug-in architecture; more details are provided in Section 2.

Thanks to external and action atoms, ACTHEX is a versatile formalism to simulate PRS. By modeling conflict resolution and fact removal strategies via external atoms, also involved realizations thereof can be elegantly captured. Furthermore, via external atoms access to heterogenous information sources in virtually any format is supported; in particular, to description logic ontologies in OWL, such that an instantiation of the OWL-PR formalism can be accommodated. Since an ACTHEX prototype implementation is available, the PRS simulation can also be effectively realized on top of it.

This work would not exist without the pioneering work of Vladimir Lifschitz on answer-set semantics. In a seminal paper together with Michael Gelfond [16], he coined what has become a predominant formalism for applied nonmonotonic reasoning.

The outline of the rest of this paper is as follows. In the next section, we briefly recall HEX and ACTHEX. We then describe how a generic PRS can be simulated in ACTHEX (Section 3), and consider possible instantiations and properties (Section 4). Related formalisms are discussed in Section 5, while Section 6 concludes the paper.

2 HEX and ACTHEX programs

2.1 HEX programs

HEX programs [11] are built over mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of *constant*, *variable*, and *external predicate names*, respectively. We follow the convention that elements of $\mathcal{X}$ (resp., $\mathcal{C}$) start with an upper-case (resp., lower-case) letter; elements of $\mathcal{G}$ are prefixed with "&". Constant names serve both as individual and predicate names. Notice, that $\mathcal{C}$ may be infinite. *Terms* are elements of $\mathcal{C} \cup \mathcal{X}$. A (*higher-order*) *atom* is of form $Y_0(Y_1, \ldots, Y_n)$, where Y_i are terms for $1 \leqslant i \leqslant n$, and $n \geqslant 0$ is the *arity* of the atom. The atom is called *ordinary*, if $Y_0 \in \mathcal{C}$. For example, $(x, type, c)$, $node(X)$, and $D(a, b)$, are atoms; the first two are ordinary atoms. Subsequently, we assume that all atoms are ordinary, i.e., of the form $p(Y_1, \ldots, Y_n)$. By $arg(a)$, $var(a)$, $pred(a)$, $arty(a)$, we understand the arguments, the variables, the predicate name, and the arity of an ordinary atom a. An *external atom* is of the form $\&g[\boldsymbol{X}](\boldsymbol{Y})$, where $\boldsymbol{X} = X_1, \ldots, X_n$ and $\boldsymbol{Y} = Y_1, \ldots, Y_m$ are lists of terms (called *input* and *output list*, resp.), and $\&g$ is an *external predicate name*. Such an atom is used to determine the truth value of an atom through an external source of computation. For example, an external atom $\&reach[arc, a](\boldsymbol{Y})$ may capture the nodes of a directed graph arc reachable from node a by querying an external computational source.

HEX-programs (or simply *programs*) are finite sets of *rules* r of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\, \beta_{n+1}, \ldots, not\, \beta_m, \tag{1}$$

where $m, k \geqslant 0$, α_i are atoms, β_j are atoms or external atoms, and "not" is *negation as failure* (or *default negation*). If $k = 0$, then r is a *constraint*; if r is variable-free, $k = 1$, and $m = 0$, then r is called a *fact*.

Furthermore, we denote by $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ the *head* of a rule r, and by $B(r) = B^+(r) \cup B^-(r)$ its *body*, where $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \ldots, \beta_m\}$ are the (sets of) *positive* and *negative body atoms*, respectively. By $cts(P)$ we understand the set of constants which appear in P.

Semantics. Answer sets of ordinary programs [16] are extended to HEX-programs P, using the FLP reduct [13]. The *Herbrand base* $\mathcal{H}_P$ of P is the set of all ground instances of atoms and external atoms occurring in P, obtained by variable substitution over $\mathcal{C}$. The grounding of a rule r, $grnd(r)$, and of P, $grnd(P) = \bigcup_{r \in P} grnd(r)$, is analogous.

An *interpretation of* P is any subset $I \subseteq \mathcal{H}_P$ containing atoms only. A satisfaction relation is defined as follows: I is a *model* of (i) an atom $a \in \mathcal{H}_P$ resp. (ii) a ground

external atom $a = \&g[\boldsymbol{x}](\boldsymbol{y})$, denoted $I \models a$, iff (i) $a \in I$ resp. (ii) $f_{\&g}(I, \boldsymbol{x}, \boldsymbol{y}) = 1$, where $f_{\&g} \colon 2^{\mathcal{H}_P} \times \mathcal{C}^{n+m} \to \{0, 1\}$ is a (fixed) *oracle function* associated with $\&g$; Intuitively, $f_{\&g}$ tells if $\boldsymbol{y}$ is in the output of the external source $\&g$ provided input $\boldsymbol{x}$. For instance, $f_{\&reach}(I, G, X, Y) = 1$ iff Y is reachable from X in the directed graph encoded by the extension of binary predicate G in I.

For a ground rule r, satisfaction is given by $I \models r$ iff $I \models H(r)$ or $I \not\models B(r)$, where (i) $I \models H(r)$ iff $I \models a$ for some $a \in H(r)$, and (ii) $I \models B(r)$ iff $I \models a$ for every $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$. As usual, I is a *model* of P, denoted $I \models P$, iff $I \models r$ for all $r \in grnd(P)$.

The *FLP-reduct* [13] of P w.r.t. an interpretation I, denoted fP^I, is the set of all $r \in grnd(P)$ such that $I \models B(r)$. Eventually, we say that $I \subseteq \mathcal{H}_P$ is an *answer set of* P, iff I is a $\subseteq$-minimal model of fP^I. The set of all answer sets of P is denoted by $\mathcal{AS}(P)$. E.g., let $P_{reach} = \{arc(1, 2); arc(2, 3); node(X) \leftarrow arc(X, Y); node(Y) \leftarrow arc(X, Y); path(X, Y) \leftarrow \&reach[arc, X](Y), node(X), X \neq Y\}$, then $\mathcal{AS}(P_{reach}) = \{A\}$, such that $\{path(1, 2), path(2, 3), path(1, 3)\} \subseteq A$.

For practical reasons, we assume that each input argument X_i of an external atom $\&g[\boldsymbol{X}](\boldsymbol{Y})$ has a type label *predicate* or *constant*, which is unique for $\&g$. Moreover, we consider external functions to be uniform in the following sense: If I and I' coincide on all the extensions of predicates x_i such that X_i is of type *predicate*, then $f_{\&g}(I, \boldsymbol{x}, \boldsymbol{y}) = f_{\&g}(I', \boldsymbol{x}, \boldsymbol{y})$. Consequently, $f_{\&g}$ depends only on the input of a given by predicate extensions and individuals.

2.2 ACTHEX programs

ACTHEX [3] is an extension of HEX programs [11] with *action atoms* built using action predicate names $\mathcal{A}$. Unlike dl-atoms in dl-programs [10], which only send and receive inputs to/from ontologies, action atoms are associated to functions capable of actually changing the state of external environments. Such atoms can appear (only) in heads of rules and as such they can be part of answer sets. An *action atom* is of the form

$$\#g\,[Y_1, \ldots Y_n]\,\{o, r\}\,[w : l]$$

where

(i) $\#g$ is an action predicate name and $Y_1, \ldots, Y_n$ is a list of terms (called *input list*), where $n = in(\#g)$ is fixed;
(ii) $o \in \{b, c, c_p\}$ is the *action option*; the action atom is called *brave* (resp., *cautious, preferred cautious*), if $o = b$ (resp., $o = c$, $o = c_p$);
(iii) r and w, the *action precedence* resp. *weight*, range over positive integers, and
(iv) l, the *action level*, ranges over variables.

For instance, an action atom $\#insert[X, Y]\{b, Pr\}$ may be defined to insert paths (pairs of nodes X, Y) into a list of paths. It is a brave atom, whose precedence is given for a respective ground instance by the instantiation of variable Pr.

An ACTHEX rule r is of the form (1) where each α_i can also be an action atom. The notion of constraint, fact etc. and the notation $H(r)$, $B(r)$ etc. naturally extends from HEX to ACTHEX rules. An ACTHEX *program* is a finite set P of ACTHEX rules, and is *ordinary*, if all its rules are ordinary, i.e. contain no external and no action atoms. As

an example consider the ACTHEX program P_{insert} consisting of the ordinary rules of P_{reach} and the rule: $\#insert[X,Y]\{b,Pr\}\leftarrow path(X,Y), Z = X * 10, Pr = Z + Y$.

Semantics. Action atoms affect an *external environment* and are executed according to *execution schedules*. Every answer set is associated with one or more *execution schedules*, which are ordered lists containing all action atoms in that particular answer set. The order of execution within a schedule depends on the actions *precedence* attribute. Action atoms allow to specify whether they have to be executed *bravely*, *cautiously* or *preferred cautiously*, respectively, meaning that the atom can get executed if it appears in at least one, all, or all *best cost* answer sets.

More formally, the Herbrand base $\mathcal{H}_P$ of an ACTHEX program P also contains action atoms (in addition to ordinary and external atoms). The grounding of a program is defined as before, and an interpretation I of P is any subset $I \subseteq \mathcal{H}_P$ containing ordinary atoms and action atoms. The satisfaction relation, and eventually the notion of answer sets, both extend straightforwardly by applying to interpretations as above, with a single slight extension:[1] for convenience, we also allow external atoms (computations) to access and take the external environment into account. In particular, we assume the external environment is represented as a finite set E of facts over a suitable language $\mathcal{L}_{env}$ We thus consider an extended oracle function $f_{\&g}\colon 2^{\mathcal{L}_{env}} \times 2^{\mathcal{H}_P} \times \mathcal{C}^{n+m} \rightarrow \{0,1\}$ associated with any ground external atom $a = \&g[\boldsymbol{x}](\boldsymbol{y})$, and define $I \models a$ iff $f_{\&g}(E, I, \boldsymbol{x}, \boldsymbol{y}) = 1$, for a given external environment $E \subseteq \mathcal{L}_{env}$. For action atoms (and ordinary atoms) a, we say that I is a *model* of a, denoted $I \models a$, iff $a \in I$.

We define the set of *best models* of P, denoted $\mathcal{BM}(P)$, as those answer sets $I \in \mathcal{AS}(P)$, where the objective function $H_P(I) = \Sigma_{i=1}^{l_{max}}(f_P(i) \times \Sigma_{a \in I \wedge l(a)=i} w(a))$ over weights and levels of action atoms in I is minimal. Here, w_{max} and l_{max} are the maximum weight and level of $ground(P)$, and f_P is defined by $f_P(1) = 1$ and $f_P(n) = f_P(n-1) \times |\{a \in ground(P) \mid w(a) \neq 0\}| \times w_{max} + 1$. Intuitively, an answer set I will be a best model if no other answer set yields a strictly lower weight on some level i, while yielding lower or equal weights than I on levels up to i. Here the weight per level is the sum of the weights of all executable actions in I with respective level.

Towards action execution we say, for a given answer set I, that a ground action $a = \#g[y_1, \ldots, y_n]\{o, r\}[w : l]$ is *executable in* I iff (i) a is brave and $a \in I$, or (ii) a is cautious and $a \in B$ for every $B \in \mathcal{AS}(P)$, or (iii) a is preferred cautious and $a \in B$ for every $B \in \mathcal{BM}(P)$. With every answer set of P we associate *execution schedules*: an execution schedule $S_P(I)$ for I is a sequence $[a_1, \ldots, a_n]$ of all actions executable in I, such that a appears before b in in the sequence if $prec(a) < prec(b)$ holds for all action atoms a and b in I. The set of all execution schedules of an answer set I of P is denoted by $\mathcal{ES}_P(I)$. Moreover, if $\mathcal{S}$ is a set of answer sets of P, then $\mathcal{ES}_P(\mathcal{S}) = \bigcup_{I \in \mathcal{S}} \mathcal{ES}_P(I)$.

For an example, observe that the only answer set A of P_{insert} contains the action atoms $\#insert[1,2]\{b,12\}$, $\#insert[2,3]\{b,23\}$, and $\#insert[1,3]\{b,13\}$, giving rise to a single execution schedule $S_{P_{insert}}(A) = [\#insert[1,2]\{b,12\}, \#insert[1,3]\{b,13\}, \#insert[2,3]\{b,23\}]$.

The *execution of an action* on an external environment E is modeled by an action function. We associate with every action predicate name #*g* an $(m+2)$-ary func-

[1] Such an extension has been suggested by Peter Schüller in a different context.

tion $f_{\#g}$ with input $(E, I, y_1, \ldots, y_m)$ that returns a new external environment $E' = f_{\#g}(E, I, y_1, \ldots, y_m)$, where $I \subseteq \mathcal{H}_P$. Based on this, given an execution schedule $S_P(I) = [a_1, \ldots, a_n]$ for I, the *execution outcome* of executing $S_P(I)$ on E, is defined as $EX(S_P(I), E) = E_n$, where $E_0 = E$, $E_{i+1} = f_{\#g}(E_i, I, y_1, \ldots, y_m)$, and a_i is of the form $\#g[y_1, \ldots, y_m]\{o, p\}[w : l]$. Intuitively the initial environment $E_0 = E$ is modified by executing every action of $S_P(I)$ in the given order, and the effect of these actions is iteratively taken into account according to the corresponding action function. Given a set $\mathcal{S}$ of answer sets of P, we use $\mathcal{EX}_P(\mathcal{S}, E)$ to denote the set of all possible execution outcomes of P on the (initial) external environment E.

For our example program P_{insert}, executing $S_{P_{insert}}(A)$ from above on an initial environment consisting of an empty list, i.e., $E_0 = []$, results in the execution outcome of an ordered list $E = [(1, 2), (1, 3), (2, 3)]$, as intended. Note that such ordered insertion can not be obtained from a HEX program without post-processing its output.

In practice, one may want only one of the possible execution outcomes of an ACTHEX program P on environment E. Either this can be achieved implicitly by appropriate modeling, such that a single (best) answer set exists, which only allows for a single execution schedule, or one has to provide corresponding selection functions explicitly.

An implementation of ACTHEX programs has been realized and is available[2] as an extension to the dlvhex system[3]. It provides simple (nondeterministic) selection functions for selecting a single answer set (the first computed among the best models) and a single execution schedule (the first computed) for it.

3 Simulating Production Rule Systems Using ACTHEX

3.1 Production Rule Systems

A Production Rule System (PRS) is in the most general case an unordered collection of conditional statements called production rules [15]. They all operate on a global database called *working memory* (WM). The left hand side of a rule (LHS) is a condition in the form of a set of positive and negative patterns, while the right hand side (RHS) contains a set of actions, typically the addition or removal of a set of facts to resp. from the WM. Sometimes, rules have priorities.

The production rule system is 'executed' in cycles, where each cycle contains the following steps:

1. *Match*: The LHS of each production rule is evaluated w.r.t. the current WM. A rule[4] for which the LHS is satisfied is called *fireable*.
2. *Conflict resolution*: From the set of fireable production rules, one is chosen according to a *conflict resolution* strategy. Such a strategy considers the priority of the rules or even the history of the execution of the system to pick a rule for execution.
3. *Act*: The actions in the RHS of the selected production rule are performed.

Such a specification of execution of a PRS is also called operational semantics.

[2] http://www.kr.tuwien.ac.at/research/systems/dlvhex/actionplugin.html

[3] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

[4] Instance of a rule, in case the rule has variables

Example 1. As an example, we describe some rules which can be used to assess a patient with chronic cough symptoms. The scenario is inspired from the step-by-step procedure described at `http://bestpractice.bmj.com/best-practice/monograph/69/diagnosis/step-by-step.html`.

As chronic cough can be caused by ACE inhibitors, in a first step the patient is asked to stop taking these medicines (in case he takes them) to check whether that is the cause. Alternatively, a set of therapeutic trials can be launched to detect whether the cough is caused by one of the common conditions: asthma, non-asthmatic eosinophilic bronchitis (NAEB), or gastro-oesophogeal reflux disease (GORD).

The scenario can be encoded by means of 4 production rules.

(1) [2]: `if` $has(P, cough)$ `and` $takes(P, acei)$ `then add` $rec(P, stop\_acei)$
(2) [2]: `if` $has(P, cough)$ `and` $has(P, wheezing)$ `then add` $treat(P, asthma)$
(3) [2]: `if` $has(P, cough)$ `then add` $treat(P, naeb)$
(4) [2]: `if` $has(P, cough)$ `and` $has(P, heartburn)$ `then add` $treat(P, gord)$

In order to monitor the state of the patient we introduce two more rules. The patient is asked how he feels: if he feels ok all symptoms are removed from the WM; otherwise, the symptom of the patient is added as a fact to the KB.

(5) [1]: `if` $ask(P, ok)$ `then remove` $has(P, X)$
(6) [1]: `if` $ask(P, X)$ `and` $\mathrm{X} \neq \mathrm{ok}$ `then add` $has(P, X)$

All rules have attached priorities: rules (1)-(4) have priority 2, while rules (5)-(6) have priority 1. If the conflict resolution strategy decides which rule should be executed solely by considering the priority of fireable rules, rules (1)-(4) will never be executed as always either the condition of (5) or the condition of (6) is fulfilled. As such, conflict resolution strategies are usually more complex: for example, a rule which has already been executed, will not be executed again until its condition is falsified and then becomes true again (i.e., only once for a particular set of bindings, where a binding is maintained until the respective condition is not fulfilled anymore). Thus, if the patient reports the same symptom, e.g. cough, several times, rule (5) will not be reexecuted: instead one of the lower priority rules (1)-(4) will be executed instead. As the lower priority rules have all the same priority, which one is actually chosen depends on satisfaction of their conditions: if more then one condition is satisfied, the rule is chosen nondeterministically (according to the intention of this example; note, however, that some systems use specificity of the condition as a further selection criterion, which would select rule (3) only if the other ones are not in the conflict set).

3.2 Production Rule Systems over HEX programs

Recently there has been an interest in augmenting Production Rules Systems with more expressive background knowledge, in the form of FOL theories, or DL KBs. The assumption is that patterns are FOL formulas which are evaluated against a KB consisting in the union of the current WM and the underlying FOL theory/DL KB [1].

In this section we introduce PRS over HEX programs. A PRS is augmented with a HEX program which together with the WM offers background knowledge. In an ASP setting, checking brave entailment of atoms, i.e. whether an atom belongs to *some* answer

set, is a common reasoning task. Actually, the possibility to express information about alternate states of the world is one of the main features of ASP. As such, the condition part of such production rules is evaluated bravely w.r.t. ASP semantics. The assumption is that for every execution cycle, some answer set A of the corresponding HEX program and current WM is randomly chosen and then all conditions are evaluated w.r.t. A.

In this setting, a *pattern* is a HEX body literal ℓ; it is *positive*, if ℓ is an atom, and *negative* otherwise. In the following, let $\mathbf{L}$ be a set atoms (called labels) such that for every $l_1, l_2 \in \mathbf{L}$: $pred(l_1) \neq pred(l_2)$ and the set $\{pred(l) \mid l \in \mathbf{L}\}$ is disjoint with $\mathcal{C}$.

Definition 1. *Let* $\mathbf{p}$ *be a finite set of patterns, let* $\mathbf{r}$*,* $\mathbf{a}$*, and* $\mathbf{e}$ *be three finite sets of atoms, and let* $l \in \mathbf{L}$ *such that* $var(l) = var(\mathbf{p}) \cup var(\mathbf{a}) \cup var(\mathbf{r}) \cup var(\mathbf{e})$*. A* HEX*-production rule (PR) is an expression of the form:*

$$[l, pr] \textit{ if } \mathbf{p} \textit{ then remove } \mathbf{r} \textit{ add } \mathbf{a} \textit{ execute } \mathbf{e} \tag{2}$$

where pr *is the priority of the rule, a natural number.*

Example 2. The encoding of rules (1)-(6) from example 1 as HEX-production rules is straightforward. For example, rule (1) can be encoded as follows:

$$[r_1(P),2] \text{ if } has(P, cough), takes(P, acei) \text{ then add } rec(P, stop\_acei).$$

We note that the presence of external atoms allows us to express more complex rules whose conditions can be external atoms with bidirectional access to external knowledge sources. For example, an external atom $\&high\_risk$ may have as inputs the extension of the predicate has and a patient, and empty output; it evaluates to `true`, iff the patient has a high risk of complications. High risk patients are referred to the emergency room:

$$[r_7(P),2] \text{ if } \&high\_risk[P, has] \text{ then add } send(P, emergency).$$

Ground instances of production rules are obtained by substitution as usual: for a term t the result of applying a substitution $\sigma : var(t) \to D$ (where D is an arbitrary set of constants) to t is denoted by $t\sigma$, and is obtained by replacing every variable in the term with its image under the substitution. This application naturally extends to sets of terms and rules. An instance of a HEX-production rule (2) is any rule

$$[l\sigma, pr] \text{ if } \mathbf{p}\sigma \text{ then remove } \mathbf{r}\sigma \text{ add } \mathbf{a}\sigma \text{ execute } \mathbf{e}\sigma \tag{3}$$

Next we define the notion of a HEX-based PRS state. The notion is intended to capture the parts of a PRS which change during a run of the system: the working memory and the history of the run itself.

Definition 2. *A* HEX*-based PRS state is a triple* $\langle WM, P, H \rangle$*, where* P *is a finite set of rules, i.e., a* HEX*-program, while* WM *and* H *are finite sets of ground facts: the working memory, and the history, respectively.*

To any set RS of production rules and a HEX-based PRS state $s = \langle WM, P, H \rangle$, we associate a *relevant domain* by $Dom(RS, s) = cts(P \cup WM \cup RS)$. We also say that a ground instance of RS is relevant w.r.t. s iff it is obtained from RS by a substition to $Dom(RS, s)$. In other words, the set of relevant ground instances of RS w.r.t. s, denoted by $rel(RS, s)$, is obtained by grounding RS over $Dom(RS, s)$.

We now turn to the firing of production rules and first define what it means for a production rule instance to be fireable in a HEX-based PRS state.

Definition 3. *Consider a set RS of* HEX*-production rules and a* HEX*-based PRS state $s = \langle P, WM, H\rangle$ such that $P \cup WM$ is consistent, i.e., it has an answer set, and let $A \in \mathcal{AS}(P \cup WM)$. A* HEX*-production rule instance of the form (3) is fireable in s w.r.t. A if and only if 1.) $A \models p'\sigma$ for all $p' \in \mathbf{p}^+$, and 2.) $A \not\models p'\sigma$ for all $p' \in \mathbf{p}^-$.*

Note that if $l\sigma$ is fireable in s, then it is also relevant w.r.t. s. As for additional notation we use $\mathit{fireable}(RS, s, A)$ to denote the set of all (relevant) instances of production rules from RS that are fireable in s w.r.t. A identified by their labels, i.e., $\mathit{fireable}(RS, s, A) = \{l\sigma \mid l\sigma \in \mathit{rel}(RS, s)$ and $l\sigma$ is fireable in s w.r.t. $A\}$.

Slightly abusing notation, we define $\mathit{fireable}(RS, s) = \mathit{fireable}(RS, s, A)$, for some $A \in \mathcal{AS}(P \cup WM)$ if $P \cup WM$ is consistent (where A is chosen nondeterministically), while $\mathit{fireable}(RS, s) = \emptyset$ otherwise.

A HEX-based PRS comprises besides a set of production rules and an initial HEX-based PRS state, also a conflict resolution function, which selects for every state of the system which PR instance should be executed next (if fireable instances exist); a change function which specifies how the working memory is affected by PR actions; and a bookkeeping function which describes how the history of a run is updated upon transition from one state to another. More specifically,

- $cr(RS_i, H)$ is the conflict resolution function, (a partial function) that given a set of PR instances RS_i and the history H, returns a single PR instance $l\sigma \in RS_i$;
- $ch(\, WM, P, \mathbf{a}, \mathbf{r}, \mathbf{e})$ is a function which given a set of ground facts representing the WM, a HEX-program P, and three sets of ground atoms, corresponding to assertion, removal, and execute actions, respectively, returns a pair $(\, WM', P')$ of a set of ground facts and a HEX-program representing the changed working memory WM' and program P' as a result of applying the actions;
- $bk(RS_i, l\sigma, H)$ is a bookkeeping function which given as input a set of PR instances RS_i, a PR instance $l\sigma$, and the history H, returns an updated history H'.

Definition 4. *A* HEX*-based PRS is a quintuple $\langle RS, s_0, cr, ch, bk\rangle$, where RS is a finite set of* HEX*-production rules of the form (2), $s_0 = \langle\, WM_0, P_0, \emptyset\rangle$ is the initial state of the system, and cr, ch, and bk are a conflict resolution, a change, and a bookkeeping function, respectively.*

We note that in this definition, a PRS may be Non-Markovian w.r.t. to the working memory, i.e., conflict resolution may select from the same conflict set RS_i produced from the same WM different rules for execution, depending on the history. This respects that standard conflict resolution functions, like forward chaining in RIF (see Section 4), take the history into account; in principle, the relevant history information may be stored in WM, and designated rules of the PRS may maintain it, such that the function $cr(RS_i, H)$ can be replaced by some function $cr'(RS_i')$ that is independent of history information H; however, bounds on the size of WM would limit the expressiveness of conflict resolution. For generality and a clean separation between data processing and rule execution management, we use the above definition.

Towards defining runs of a HEX-based PRS in terms of transition functions, let us first consider single transitions. A triple $(s, l\sigma, s')$, where $s = \langle\, WM, P, H\rangle$ and $s' = \langle\, WM', P', H'\rangle$ are HEX-based PRS states and $l\sigma$ is the label of a PR instance

of the form 3, is a *valid transition* of a PRS $PS = \langle RS, s_0, cr, ch, bk\rangle$ iff (i) $l\sigma = cr(\mathit{fireable}(RS, s), H)$, (ii) $(WM', P') = ch(WM, P, \mathbf{a}, \mathbf{r}, \mathbf{e})$, and (iii) $H' = bk(\mathit{fireable}(RS, s), l\sigma, H)$.

We call a state s *reachable* in (a run of) a PRS PS iff either $s = s_0$ is the initial state of PS, or there exists a valid transition $(s', l\sigma, s)$ such that s' is reachable. Furthermore, we denote by S the set of all states and by $L\Sigma$ the set of all labels of PR instances.

Definition 5. *Let $PS = \langle RS, s_0, cr, ch, bk\rangle$ be a* HEX*-based PRS. A relation $\rightarrow_{PS} \subseteq S \times L\Sigma \times S$ is called a* run *of PS iff $(s, l\sigma, s') \in \rightarrow_{PS}$ implies that $(s, l\sigma, s')$ is a valid transition such that s is reachable.*

Moreover, if there exists $(s, l\sigma, s_f) \in \rightarrow_{PS}$ such that $(s_f, l'\sigma', s') \notin \rightarrow_{PS}$ for all $l'\sigma' \in L\Sigma$ and $s' \in S$, then the run is *finite* (in other words, the run *terminates*), otherwise the run is *infinite*.

3.3 Simulating HEX-based PRSs in ACTHEX

In this section we show how runs of HEX-based PRSs can be simulated via a translation of such systems to ACTHEX programs. The translation reenacts the operational semantics of HEX-based PRSs in terms of execution schedules of the corresponding ACTHEX programs. While ACTHEX programs do not have an explicit notion of state, they allow for stateful computation via the external environment: a stateful program thus has all state dependent information as a part of the external environment, which is subject to be updated and accessed via action atoms and external atoms.

In our particular case, simulating the behavior of a HEX-based PRS, the state dependent information consists of the WM, the HEX program, and the history of the current run. The facts of the WM and the HEX program rules are part of the actual ACTHEX program which is to be executed for simulation of the PRS behavior. As such, the ACTHEX program is dynamic, i.e., subject to change—the PRS execution cycle is simulated by repeated executions of suitable modifications of the original program (which address just the facts representing the working memory and the HEX program rules). Technically, this is achieved via an action atom $\#execute$ which appears as the last action in every non-empty execution schedule of the ACTHEX program. Its execution results in a recursive call for evaluating the modified program (with updated WM facts and HEX rules).

For a more formal account, consider a HEX-based PRS $PS = \langle RS, s_0, cr, ch, bk\rangle$. Below, we will first give the encoding of the static part of the ACTHEX program which represents (and eventually simulates) the production rules RS. This fixed part of the ACTHEX program will be denoted by Π_{RS}. Furthermore, we will use indices $i \geqslant 0$ to denote the (initial) external environment E_i of iterative evaluations of the ACTHEX program. More specifically we will use E_i^{WM}, E_i^{HEX}, and E_i^H to refer to the different parts of E_i, representing corresponding state information. In slight abuse of notation, we will also use E_i^{WM} and E_i^{HEX} to denote the set of facts and the set of rules that together with Π_{RS} constitute the ACTHEX program to be evaluated (executed) at step i.

There are several issues of concern. To simulate an execution cycle, one has to (i) capture rule fireability conditions, (ii) select a rule for execution (conflict resolution), (iii) execute the selected rule, and (iv) update the history. After that, one has to (v) move on to the next execution cycle.

(i) The simulation of pattern matching in ACTHEX is straightforward: production rule conditions are captured by bodies of ACTHEX rules: let m be the maximum arity of labels of rules in RS, let *fires* be an ACTHEX predicate of arity $m + 4$. For every production rule of the form (2), we add the following rule to Π_{RS}:

$$\mathit{fires}(\mathit{pred}(l), \mathit{arity}(l), \mathit{arg}(l), pr, 0, \ldots 0) \leftarrow p,$$

where the argument part of *fires* is padded with zeroes up to $m+4$ arguments.[5] Intuitively, $\mathit{fires}(\mathit{pred}(l), \mathit{arity}(l), \mathit{arg}(l), pr, 0, \ldots 0)$ is satisfied in an answer set of $\Pi_{RS} \cup E_i^{WM} \cup E_i^{\text{HEX}}$ iff there exists a fireable instance $l\sigma$ of the production rule with label l w.r.t. an answer set of $E_i^{WM} \cup E_i^{\text{HEX}}$.

(ii) The conflict resolution strategy is outsourced to an external atom $\&\mathit{cres}[\mathit{fires}](X_1, \ldots, X_{m+2})$. Its external function $f_{\&\mathit{cres}}$ takes the extension of *fires* in the given interpretation I into account as well as the history of the run E_i^H in the environment to select a production rule instance for output. As for *fires*, we use reification to describe a rule instance; hence the output of $\&\mathit{cres}$ has arity $m+2$, where m is again the maximum arity of a label. It represents $\mathit{cr}(\mathit{fireable}(RS, \langle E_i^{WM}, E_i^{\text{HEX}}, E_i^H \rangle, E_i^H)$.

(iii) Once a rule instance is selected, its action part should be executed. For every production rule of form (2) where $\mathbf{a} = \cup_{1 \leqslant i \leqslant n} a_i$, $\mathbf{r} = \cup_{1 \leqslant i \leqslant p} r_i$, and $\mathbf{e} = \cup_{1 \leqslant i \leqslant q} e_i$, let

$$\#\mathit{action}[a_1, \mathit{arity}(a_1), \mathit{arg}(a_1), \ldots, \\ r_1, \mathit{arity}(r_1), \mathit{arg}(r_1), \ldots, e_1, \mathit{arity}(e_1), \mathit{arg}(e_1), \ldots, 0, \ldots, 0]\{b, 1\}$$

be an action atom whose execution is intended to simulate the effect of the execution of the production rule instance: it changes the working memory, and the HEX program, such that their new contents are exactly those given by $ch(E_i^{WM}, E_i^{\text{HEX}}, \mathbf{a}, \mathbf{r}, \mathbf{e})$. Note that the mode of execution for such an action is *brave* and its priority is 1. For simplicity, we will refer in the following to such an atom as act_l. The arity of $\#\mathit{action}$ is $max_{l \in RS}(\sum_{a_i \in \mathbf{a}}(|\mathit{arg}(a_i)| + 2) + \sum_{r_i \in \mathbf{r}}(|\mathit{arg}(r_i)| + 2) + \sum_{e_i \in \mathbf{e}}(|\mathit{arg}(e_i)| + 2))$.

For every production rule with label l, we also add the following rule to Π_{RS}:

$$\mathit{act}_l \leftarrow \&\mathit{cres}[\mathit{fires}](\mathit{pred}(l), \mathit{arity}(l), \mathit{arg}(l), 0, \ldots, 0).$$

(iv) After updating the WM and the HEX program, the history of the run is updated by means of an action atom $\#\mathit{update}$ whose effect is to change the history to the one returned by $bk(\mathit{fireable}(RS, \langle E_i^{WM}, E_i^{\text{HEX}}, E_i^H \rangle), l\sigma, E_i^H)$. It takes the extension of *fires* in I and the input rule instance $l\sigma$ (in reified form; from $\&\mathit{cres}$) into account.

$$\#\mathit{update}[\mathit{fires}, X_1, \ldots, X_{m+2}]\{b, 2\} \leftarrow \&\mathit{cres}[\mathit{fires}](X_1, \ldots, X_{m+2}).$$

(v) Finally, the next execution cycle is triggered by means of an action atom $\#\mathit{execute}$: its effect is to build and execute the ACTHEX program $\Pi_{RS} \cup E_{i+1}^{WM} \cup E_{i+1}^{\text{HEX}}$. Again, this happens only if a production rule action has actually been executed (which means that the external environment has already been changed to contain E_{i+1}^{WM} and E_{i+1}^{HEX} and can thus be used to build the ACTHEX program for the next step).

$$\#\mathit{execute}\{b, 3\} \leftarrow \&\mathit{cres}[\mathit{fires}](X_1, X_2, \ldots, X_{m+2}).$$

Note that whenever an execution schedule $S_P(I)$ of an ACTHEX program P contains an action atom $\#\mathit{execute}$, whose effect is the execution of another ACTHEX program P',

[5] Here, for ease of exposition, we use reification to encode a production rule instance as an argument of an action atom. Regarding efficiency, this is suboptimal: rather than padding, one could use different predicates $\#fires_k$ with fixed arity k, for $1 \leqslant k \leqslant m + 4$.

then $S_P(I)$ can be regarded as being interleaved with an execution schedule $S_{P'}(I')$ of P'. Intuitively, $\#execute$ in $S_P(I)$ is replaced with $S_{P'}(I')$. If $S_{P'}(I')$ in turn contains an $\#execute$ atom, and so forth, then $S_P(I)$ is essentially infinite.

In the following let $act_{l\sigma}$ denote $\#action[a_1, arity(a_1), arg(a_1)\sigma, \ldots, r_1, arity(r_1), arg(r_1)\sigma, \ldots, e_1, arity(e_1), arg(e_1)\sigma, \ldots, 0, \ldots, 0]$, and let $\#update_{l\sigma}$ denote $\#update[fires, pred(l), arity(l), arg(l)\sigma, 0, \ldots, 0]$.

Lemma 1. *Given a* HEX*-based PRS* $PS = \langle RS, s_0, cr, ch, bk\rangle$ *such that* $s_0 = \langle WM_0, P_0, \emptyset\rangle$*, let* $E_0^{WM} = WM_0$*,* $E_0^{\text{HEX}} = P_0$*,* $E_0^H = \emptyset$*, and let* $\Pi = \Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$ *be an* ACTHEX *program. Then, an execution schedule* S_Π *of* Π *is either empty or has the form:* $[\#act_{l_1\sigma_1}, \#update_{l_1\sigma_1}, \#act_{l_2\sigma_2}, \#update_{l_2\sigma_2}, \ldots,]$*. Moreover, the range of* σ_i *is given by* $cts(E_{i-1}^{WM} \cup E_{i-1}^{\text{HEX}})$*, for* $i > 1$*.*

The following proposition shows the soundness of the translation: every execution schedule of an ACTHEX program $\Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$ corresponds to a run of PS.

Proposition 1. *Let* $PS = \langle RS, s_0, cr, ch, bk\rangle$ *be a* HEX*-based PRS such that* $s_0 = \langle WM_0, P_0, \emptyset\rangle$*. For every execution schedule* S_Π *of* $\Pi = \Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$ *and corresponding external environments sequence* $(E_i)_{i\geqslant 0}$*, where* $E_i = \langle E_i^{WM}, E_i^{\text{HEX}}, E_i^H\rangle$*, there exists a run* $\rightarrow_{\text{HEX}-PRS}$ *of* PS *such that the following holds for every* $k \geqslant 1$*:*

- *if* $\#act_{l_k\sigma_k}$ *is in* S_Π*, then there exists* $(s, l\sigma, s') \in \rightarrow_{\text{HEX}-PRS}$ *such that* $E_{k-1} = s$*,* $E_k = s'$*,* $l_k = l$*, and* $\sigma_k = \sigma$*.*

Moreover, the simulation is also complete: for every run of a PRS PS, there exists a counterpart execution schedule of the ACTHEX program $\Pi = \Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$.

Proposition 2. *Let* $PS = \langle RS, s_0, cr, ch, bk\rangle$ *be a* HEX*-based PRS such that* $s_0 = \langle WM_0, P_0, \emptyset\rangle$*. For every run* $\rightarrow_{\text{HEX}-PRS}$ *of* PS *there exists an execution schedule* S_Π *of* $\Pi = \Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$ *and a corresponding external environments sequence* $(E_i)_{i\geqslant 0}$*, where* $E_i = \langle E_i^{WM}, E_i^{\text{HEX}}, E_i^H\rangle$*, such that the folloing holds:*

- *for every* $(s, l\sigma, s') \in \rightarrow_{\text{HEX}-PRS}$ *there exists an integer* $k \geqslant 1$ *such that* $s = E_{k-1}$*,* $s' = E_k$*,* $l = l_k$*, and* $\sigma = \sigma_k$*.*

From Propositions 1 and 2, we easily obtain the following:

Corollary 1. *There exists a terminating run of a PRS* PS *iff there exists a finite execution schedule for* $\Pi_{RS} \cup E_0^{WM} \cup E_0^{\text{HEX}}$*.*

4 Instantiations and Properties

In Section 3, we provided a general framework for PRSs over HEX programs, in which the conflict resolution strategy, the effects of actions executions, and the history update were generic parameters of a PRS. We now discuss possible ways to instantiate these parameters. As the notions of history and conflict resolution strategy are tightly related, (the strategy uses the history to select a rule for execution), we will treat them together.

Conflict Resolution Strategy/History. In general, a CRS chooses one of the fireable rules for execution. The complexity of a strategy can vary from a simple nondeterministic selection to a multi-tier selection mechanism, as the one given by the `rif:forwardChaining` strategy in RIF-PRD [25]. It uses history related information like the first time a rule instance was in the set of fireable rules, and the last time it did not fire. More specifically, the following rules are applied to select one fireable rule in decreasing order of priority:

- *refraction*: a rule instance which has been previously selected must not be selected again if the reasons that made it eligible for firing in the first place still hold;
- *priority*: only rule instances with the highest priority are maintained;
- *recency*: rule instances are ordered by the number of consecutive system states in which they are fireable, only the most recently fireable ones are eligible for firing;
- for the remaining rules, break the tie somehow.

We discuss how to encode parts of this strategy in ACTHEX. First the $\&update$ action can be detailed such that it stores in `hist` precisely the information needed by the strategy. We assume that `hist` contains for every rule instance $l\sigma$ two counters similar to the $recency$ and $lastPicked$ counters described in the RIF-PRD specification, which indicate the number of consecutive times $l\sigma$ was in the set of fireable rules, and the last time $l\sigma$ fired, respectively.

Next we show how to encode a refraction and priority-based selection in ACTHEX. Let $\&refr$ be an external atom which uses the counters stored in `hist` to indicate refracted rules: the atom $\&refr(pred(l), arity(l), arg(l), pr, 0, \ldots, 0)$ evaluates to true iff a production rule instance $l\sigma$ is refracted[6]. To capture the sets of refracted and nonrefracted rule instances, we use for every production rule with label l and priority pr two ACTHEX rules as follows:

$$refr(pred(l), arity(l), arg(l), pr, 0, \ldots 0) \leftarrow fires(pred(l), arity(l), arg(l), pr, 0, \ldots 0), \\ \&refr[pred(l), arity(l), arg(l), pr, 0, \ldots 0)]()$$

$$ref(pred(l), arity(l), arg(l), pr, 0, \ldots 0) \leftarrow fires(pred(l), arity(l), arg(l), pr, 0, \ldots 0), \\ not\ \&refr[pred(l), arity(l), arg(l), pr, 0, \ldots 0)]()$$

The selection among the nonrefracted rules of only those highest priority can be done without accessing the history. For every pair (l, l') of production rules, where l has priority pr and l' has priority pr', we introduce the following ACTHEX rule:

$$spr(pred(l), arity(l), arg(l), pr, 0, \ldots 0) \leftarrow ref(pred(l), arity(l), arg(l), pr, 0, \ldots 0), \\ \&fires(pred(l'), arity(l'), arg(l'), pr', 0, \ldots 0), pr > pr'$$

Finally, top priority rules (i.e., those kept in) are those not having small priorities:

$$tpr(pred(l), arity(l), arg(l), pr, 0, \ldots 0) \leftarrow ref(pred(l), arity(l), arg(l), pr, 0, \ldots 0), \\ not\ spr(pred(l), arity(l), arg(l), pr, 0, \ldots 0)$$

We note that the last two steps of `rif:forwardChaining`, *recency* and nondeterministic choice, can be encoded similarly: for the former we must access `hist` using an external atom, while the latter can be encoded by non-deterministic selection.

[6] For more information how the counters determine whether a rule is refracted or not please consult the RIF-PRD specification.

Change. Another benefit in terms of versatility of HEX-based PRS and our realization using ACTHEX, is enhanced flexibility of modeling and representing change. A plain PR style instantiation of the function $change(\,WM\,, P, \mathbf{a}\sigma, \mathbf{r}\sigma, \mathbf{e}\sigma)$ would execute all of $\mathbf{e}\sigma$ on WM without changing the working memory and return $WM' = (\,WM \setminus \mathbf{r}\sigma) \cup \mathbf{a}\sigma$.

HEX-based PRS enable richer changes to be realized in a straightforward and often declarative way. By simple encoding techniques, atoms in PR rule heads can represent more sophisticated changes to the working memory. E.g., an atom $remove\_ext(X)$ whose ground instances (where a constant c replaces X), represent the removal of the entire extension of a predicate named c from WM; such predicate retraction has been proposed as an advanced feature in RIF-PRD (see also next section).

By augmenting a PRS with a declarative component such as HEX-rules, more subtle interactions such as logical ramification may have to be respected when updating the working memory. Recalling the simple example from the introduction, if a fact A is to be removed, the presence of another fact B and a rule $B \rightarrow A$ in the background still allows to derive A. Problems of this nature have been studied extensively in AI as belief revision and contraction problems (cf. [22] for an overview). Respective solutions and techniques (such as WIDTIO used e.g. in [24]) can be applied and realized via instantiations of $change$, which also has access to the HEX-program P.

Eventually, the augmentation with HEX-programs offers a simple means to create new objects using external atoms. While desirable in modifying the working memory for state of the art PRS, this feature must be used carefully to ensure termination.

Termination. One aspect of encoding PRS in ACTHEX is that in general, there is no guarantee that ACTHEX programs which have recursive evaluation calls using $execute$ terminate. In fact, properties of a PRS like termination, or whether a certain fact holds in some execution/all runs, etc. are not decidable in general, and thus analog properties of the ACTHEX encoding are necessarily undecidable in general. However, under suitable restrictions they are decidable, and consequently the respective properties of PRS; studying such restrictions, guided by [27], remains for future work.

Interfacing with External Sources. Obviously, augmenting PRS with HEX benefits from the fundamental aim of HEX-programs: providing a declarative interface to external information sources and software. To date, ontologies are a common means of representing terminological (background) knowledge that systems can process autonomously and utilize to provide domain specific solutions for various application scenarios. Interfacing an ontology via an external atom is a premier use case of HEX-programs (especially DL-programs [10]). In our example, rather than hard-coding recommendations associated with symptoms in rules, an external medical ontology might be accessed for classifying symptoms and relating them to suitable recommendations, which the PRs then execute.

Clearly, ontologies are only one type of external information that might be relevant for building applications. Through external atoms, HEX-based PRS can incorporate various heterogeneous external sources such as calendars, various databases etc.

5 Discussion

We now briefly address the potential of the ACTHEX simulation w.r.t. RIF, and consider some alternative formalisms using answer set semantics for PRS simulation.

5.1 RIF Potential

The Rule Interchange Format (RIF) working group developed a suite of W3C recommended rule formalisms, including combinations with RDF and OWL.

RIF's production rule dialect, RIF-PRD, is a standard XML serialization format for production rule languages serving as a lingua franca for rule exchange. Roughly, it hosts rules with atomic actions, like *assert fact*, *retract fact*, *retract all slot values*, *retract object*, *execute*, and *compound* actions, like *modify fact*. The action (then) part of a rule is a sequence of actions preceded by action variable declaration patterns; by the *frame object declaration* pattern, new individuals can be introduced as a side effect of executing an assert action. The condition (if) part is a condition formula, built from atomic formulas using Boolean connectives and existential quantification. In normalized rules, conditions are disjunction-free and compound actions replaced by sequences of atomic actions. Rules can be put in groups, for which priorities and CRS can be defined.

The semantics of atomic actions is specified by the RIF-PRD transition relation: $\rightarrow_{RIF-PRD} \subseteq W \times L \times W$, where W consists of all states of the fact base and L of all ground atomic actions; notably, retraction must respect subclass and class instance relationships in the facts (which are easy to express in HEX background).

RIF-PRD defines PRS semantics in terms of a labeled transition relation $\rightarrow_{PRS} \subseteq S \times L \times S$ on system states S, where the labels L are sequences of ground action atoms and $(s, a, s') \in \rightarrow_{PRS}$ means that $(facts(s), a, facts(s'))$ is in the transitive closure of $\rightarrow_{RIF-PRD}$ and $a = actions(picked(s))$ is from the firing rules selected by the CRS. As default CRS, RIF-PRD defines `RIF:forwardChaining`, but others can be used.

The PRS simulation in ACTHEX outlined above provides a basis for realizing instances of RIF-PRD, where in particular external atoms and actions atoms are helpful to instantiate generic elements of the description, such as $picked(\cdot)$, $actions(\cdot)$ and realization of the transition relation (i.e., realizing $assert(\phi)$, $retract(\phi)$ etc). However, a detailed discussion of how to realize the complex standard is beyond this paper; we just note that answer set semantics has been used to simulate RIF-PRD before (see below).

An important aspect of using ACTHEX programs is that external atoms enable combinations with other formats. Marano et al. [20] showed how to simulate RIF-Core rules combined with OWL2RL ontologies using HEX programs, by casting the ontologies to RIF-Core; in the same vein, RIF-PRD plus OWL2RL may be realized using ACTHEX. Due to the expressiveness of answer set semantics, combinations with other OWL profiles and RDF may be done under suitable assumptions (cf. [4, 26]). Finally, a loose-coupling semantics of RIF-OWL à la [10] can easily forged from the ACTHEX simulation.

5.2 Alternative Approaches based on Answer Sets

$\mathbb{FDNC}$**.** Natural candidates are answer set based formalisms which can express a forward notion of time and have well-studied decidable reasoning tasks. Such a formalism

are $\mathbb{FDNC}$ programs [12], which are a fragment of ASP with function symbols that achieves decidability via the forest model property, but allows only for unary and binary predicates and restricts the rule syntax. Arbitrary predicate arity is supported in *higher-arity* $\mathbb{FDNC}$, which imposes syntactical restrictions on variables usage are imposed to maintain decidability. Algorithms for standard reasoning tasks like deciding program consistency, cautious/brave entailment of atoms, etc. are available. In [12], a translation of the action language $\mathcal{K}$ (which was inspired by the action language $\mathcal{C}$ [18]) to $\mathbb{FDNC}$ is provided. Using a similar translation, potentially infinite PRS runs could be simulated; the reasoning support for $\mathbb{FDNC}$ allows then to check static properties of an encoded PRS. However, due to lacking actions in $\mathbb{FDNC}$, the results of execution runs cannot be materialized. Moreover, the lack of external atoms prevents loosely-coupled interaction with external sources. It remains to explore how $\mathbb{FDNC}$ can be extended with external atoms that allow sending inputs, querying, and modifying external sources.

STLP. Another answer-set based formalism that can simulate a forward time line are Splittable Temporal Logic Programs (STLP) [6]. They are a fragment of Temporal Equilibrium Logic, an extension of ASP with modal temporal operators. An algorithm for reasoning with such programs is provided in [6]; temporal equilibrium models are captured by an LTL formula using two well-known techniques in ASP: program splitting and loop formulas. The algorithm has been implemented using the LTL model checker SPOT [7]. Similar considerations as for $\mathbb{FDNC}$ apply for PRS encodings using STLP.

Incremental ASP. Damasio et al. [8] used ASP to realize a simulation of the default semantics of RIF-PRD. To this end, they described an encoding of RIF-PRD into the incremental ASP solver iClingo,[7] in which roughly a program can be incrementally evaluated, in a stateful manner, where the current program slice is instantiated w.r.t. the current increment value, the already evaluated program slices into account. This mechanism is particularly attractive to generate a (finite) trajectory for the execution of a sequence of (possibly nondeterministic) actions. In particular, [8] presents a nice encoding of the `RIF:forwardChaining` CSR in iClingo. The encoding produces as a result an answer set of the program, which describes an execution run; the real execution, however, must be accomplished separately. In our ACTHEX encoding, action execution is an integral part. In addition, iClingo does not provide access to external sources, nor to an external environment; thus, coupling RIF-PRD with ontologies and linked data, and access to other data sources is unsupported and requires further work.

6 Conclusion

We have discussed how production rule systems (PRS) can be encoded in answer set programs with external source access, and in particular how they can be simulated using ACTHEX programs, which extend HEX programs with actions. Thanks to its generic interfacing and plugin architecture, ACTHEX allows to realize a range of conflict resolution and change strategies, and in addition to access and combine PRS with heterogeneous data sources, like ontologies, RDF stores, thesauri etc.; e.g., external atoms

[7] `http://potassco.sourceforge.net/#iclingo`

make loose coupling with description logic ontologies easy, but also tight coupling as envisaged by RIF may be hosted, extending work of [8, 20]. In fact, the formalism offers a smooth integration of three worlds: production rules, logical rules, and ontologies.

Several issues remain for future work. While we have described simulation of PRS, we did not discuss issues like termination and reasoning over ACTHEX programs encoding PRS, nor computational complexity. A detailed study of termination and complexity, guided by [27], is necessary. The general setting of our simulation, in which generic components must be instantiated and also the environment can be changed, covers a large space of concrete settings, and identifying the most relevant ones will be important. With regard to reasoning, it would be interesting to consider properties expressed in temporal logic similar as in STLP [6] and to extend the algorithm and techniques there to suitable settings for ACTHEX programs. Finally, an implementation of the generic PRS simulation as a front end to the ACTHEX prototype remains to be done. In the course of this, also libraries for conflict resolution and change strategies should be built.

Acknowledgement. We are grateful to Jim Delgrande for useful comments and suggestions which helped to improve this work.

References

1. Rezk, M., Nutt, W.: Combining Production Systems and Ontologies. In: The Fifth International Conference on Web Reasoning and Rule Systems (2011)
2. Baral, C., Lobo, J.: Characterizing production systems using logic programming and situation calculus, `www.cs.utep.edu/baral/papers/char-prod-systems.ps`
3. Basol, S., Erdem, O., Fink, M., Ianni, G.: HEX programs with action atoms. In: Tech. Comm. of ICLP 2010. Leibniz International Proc. in Informatics (LIPIcs), vol. 7, pp. 24–33 (2010)
4. de Bruijn, J., Pearce, D., Polleres, A., Valverde, A.: Quantified equilibrium logic and hybrid rules. In: Marchiori, M., et al. (eds.) Proc. RR-2007. LNCS 4524, pp. 58–72. Springer (2007)
5. de Bruijn, J., Rezk, M.: A logic based approach to the static analysis of production systems. In: Polleres and Swift [23], pp. 254–268
6. Cabalar, P.: Loop formulas for splitable temporal logic programs. In: Delgrande, J., Faber, W. (eds.) Proc. LPNMR-2011. LNCS 6645, pp. 80–92. Springer (2011)
7. Cabalar, P., Diguez, M.: Stelp – a tool for temporal answer set programming. In: Delgrande, J., Faber, W. (eds.) Proc. LPNMR-2011. LNCS 6645, pp. 370–375. Springer (2011)
8. Damásio, C.V., Alferes, J.J., Leite, J.: Declarative semantics for the rule interchange format production rule dialect. In: Patel-Schneider, P.F., et al. (eds.) Pro ISWC-2010 (1). LNCS 6496, pp. 798–813. Springer (2010)
9. de Bruijn, J., Bonnard, P., Citeau, H., Dehors, S., Heymans, S., Pührer, J., Eiter, T.: Combinations of rules and ontologies: State-of-the-art survey of issues. Tech. Rep. Ontorule D3.1, Ontorule Project Consortium (2009), `http://ontorule-project.eu/`
10. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. Artif. Intell. 172(12/13), 1495–1539 (2008)
11. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: Proc. IJCAI-2005. pp. 90–96. Professional Book Center (2005)
12. Eiter, T., Šimkus, M.: $\mathbb{FDNC}$: Decidable non-monotonic disjunctive logic programs with function symbols. ACM Trans. Computational Logic (TOCL) 11(2), article 14

13. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1), 278–298 (2011)
14. Feier, C., Aït-Kaci, H., Angele, J., de Bruijn, J., Citeau, H., Eiter, T., Ghali, A.E., Kerhet, V., Kiss, E., Korf, R., Krekeler, T., Krennwallner, T., Heymans, S., (FUB), A.M., Rezk, M., Xiao, G.: Complexity and optimization of combinations of rules and ontologies. Tech. Rep. Ontorule D3.3, Ontorule Project Consortium (2010) http://ontorule-project.eu/
15. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. 19(1), 17–37 (1982)
16. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
17. Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. AI 2(3-4), 193–210 (1998)
18. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic Causal Theories. Artif. Intell. 153(1-2), 49–104 (2004)
19. Kowalski, R.A., Sadri, F.: Integrating logic programming and production systems in abductive logic programming agents. In: Polleres and Swift [23], pp. 1–23
20. Marano, M., Obermeier, P., Polleres, A.: Processing RIF and OWL2RL within DLVHEX. In: Hitzler, P., Lukasiewicz, T. (eds.) Proc. RR-2010. LNCS 6333, pp. 244–250. Springer (2010)
21. Motik, B., Rosati, R.: Reconciling Description Logics and Rules. JACM 57(5), 1–62 (2010)
22. Peppas, P.: Belief revision. In: van Harmelen, F., et al. (eds.) Handbook of Logic in AI and Logic Programming, chap. 8, pp. 317–360. Elsevier (2008)
23. Polleres, A., Swift, T. (eds.): Proc. Int'l Conf. on Web Reasoning and Rule Systems (RR) 2009, LNCS 5837. Springer (2009)
24. Rezk, M., Kifer, M.: Formalizing production systems with rule-based ontolgies. In: Lukasiewicz, T., Sali, A. (eds.) Proc. FoIKS-2012, LNCS, Springer (2012) to appear
25. de Sainte Marie, C., Hallmark, G., Paschke, A. (eds): RIF Production Rule Dialect. Recommendation 22 June 2010, W3C (2010), http://www.w3.org/TR/rif-prd/
26. Rosati, R.: $\mathcal{DL}$+LOG: Tight integration of description logics and disjunctive Datalog. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. KR-2006. pp. 68–78. AAAI Press (2006)
27. Rosati, R., Franconi, E.: Generalized ontology-based production systems. In: Proc. KR-2012. AAAI Press (2012), to appear
28. Waterman, D., Hayes-Roth, F.: Pattern-directed inference systems. Academic Press (1978)