A Logical Characterization of a Reactive System Language

Robert Kowalski and Fariba Sadri

Department of Computing
Imperial College London
{rak,fs@doc.ic.ac.uk}

Abstract. Typical reactive system languages are programmed by means of rules of the form *if antecedent then consequent*. However, despite their seemingly logical character, hardly any reactive system languages give such rules a logical interpretation. In this paper, we investigate a simplified reactive system language KELPS, in which rules are universally quantified material implications, and computation attempts to generate a model that makes the rules *true*.

The operational semantics of KELPS is similar to that of other reactive system languages, and is similarly incomplete. It cannot make a *rule* true by making its antecedent *false*, or by making its consequent *true* whether or not its antecedent becomes *true*. In this paper, we characterize the *reactive models* computed by the operational semantics. Informally speaking, a model is *reactive* if every action in the model is an instance of an action in the consequent of a rule whose earlier conditions are *true*.

Keywords: reactive systems, model generation, completeness, LPS, KELPS.

1 Introduction

State transition systems play an important role in many areas of Computing. They underpin the operational semantics of imperative programming languages, the dynamic behavior of database management systems, and many aspects of knowledge representation in artificial intelligence. In many of these systems, state transitions are performed by executing reactive rules, which describe relationships between earlier and later states. Reactive rules occur explicitly as condition-action rules in production systems, event-condition-action rules in active databases, and transition rules in Abstract State Machines [4]. They are implicit in Statecharts [5] and BDI agents plans. They are the core of Reaction RuleML [16].

Simple state transition systems have an operational semantics in which computation consists in generating a finite sequence $act_1, \ldots, act_n$ of actions to transform an initial state S_0 into a goal state S_n. In this paper, we investigate the logical semantics of a state transition system, KELPS, which generates actions by using reactive rules. Given an initial state S_0 and a potentially non-terminating sequence of external events $ext_1,\ldots, ext_i,\ldots$, computation in KELPS consists in generating associated sequences

$act_1, \ldots, act_i, \ldots$ of actions and states $S_1, \ldots, S_i, \ldots$, with the purpose of making the reactive rules *true*.

In previous papers [11, 12, 14], we presented a Logic-based agent and Production System language LPS, which combines reactive rules with logic programs. KELPS [13] is a simplified Kernel of LPS without logic programs. Its operational semantics is similar to that of imperative reactive rule languages, which generate sequences of actions and states, but maintain only a single current state, using destructive state transformations. However, the reactive rules in KELPS are represented in first-order logic (FOL), as sentences of the form $\forall X$ [*antecedent*$\rightarrow \exists Y$ [*consequent*]], where all the time-stamps in *consequent* are constrained to be later than or equal to the latest time in *antecedent*

KELPS is not intended to be a practical computer language, but has been simplified to focus on the main issues concerning the logical semantics of reactive system languages more generally. It can be regarded as a compiled form of LPS, in which relations defined intensionally by logic programs are compiled into extensionally defined predicates, defined by atomic sentences. The resulting kernel language is not as expressive and well-structured as LPS, but is still very expressive compared with many other reactive system languages. In particular, antecedents of rules in KELPS can recognize a large class of complex events, and consequents of rules can generate complex alternative, conditional plans of actions.

In [11, 12, 14] we showed that the operational semantics (OS) of LPS (and therefore of KELPS) is *sound*: Any sequence of states and events that the OS recognizes as solving the computational task is indeed a model that makes the reactive rules *true*. However, the OS is *incomplete*, because it generates only *reactive models*, in which the consequents of reactive rules are made *true* after their antecedents become *true*. It does not generate models that *proactively* make consequents *true* whether or not their antecedents become *true*; that make antecedents *false*, to prevent making their consequents *true*; or that unnecessarily make their antecedents *true*, and are then forced to make their consequents *true*. Moreover, it does not generate models that contain actions that are irrelevant to the computational task.[1]

Fig. 1 presents an informal illustration of the different kinds of models. In addition to a single reactive rule, the example also includes a causal theory, used to update states destructively, and a definition of the temporal constraint predicates, defined extensionally by means of atomic sentences. All of the models include both the set of all external events and the set of all actions motivated by the reactive rule and triggered by the external events. The non-reactive models also contain additional, unmotivated or unnecessary actions.

In this paper, we characterize the reactive models ***I*** generated by the KELPS OS. These models all have the property that every action in ***I*** is motivated by being an instance of an action that occurs explicitly in the consequent of a reactive rule whose earlier conditions are already *true*. For example, in the reactive model of Fig. 1, the action *dispatch(bob, book, tuesday)*, is an instance of *dispatch(C, Item, T2)* in the consequent of the rule, and all earlier conditions, *orders(bob, book, monday)* and

[1] In section 6.1, we discuss the relationship between these different kinds of models and the models that are generated using abductive logic programming [7] and the frame axioms of the event calculus [15]. We will see that reactive models need not be minimal.

reliable(bob, monday), of the associated instance of the rule are *true* before the time of the action. However, in the proactive model, although the action *dispatch(mary, book, tuesday)* is also an instance of *dispatch(C, Item, T2)*, the earlier instance *reliable(mary, monday)* of the condition *reliable(C, T1)* is not *true*. Moreover, in the irrelevant model, the action *send-voucher(mary, wednesday)* has no relationship with any of the actions in the rule at all.

In the remainder of the paper, we present the KELPS language (section 2), its model-theoretic (section 3) and operational semantics (section 4), the relationship between the two semantics (section 5), the relationship with Abductive Logic Programming [8], MetateM [1] and Transaction Logic [2] (section 6), and future work (section 7).

If a customer orders an item and the customer is reliable,
then dispatch the item and send an invoice to the customer for the item.

Reactive rule: $orders(C, Item, T1) \wedge reliable(C, T1)$
$\rightarrow dispatch(C, Item, T2) \wedge send\text{-}invoice(C, Item, T3) \wedge$
$T1 < T2 \leq T3 \leq T1 + 3$

Auxiliary predicate definitions: *monday < tuesday, tuesday < wednesday, etc.*

Causal theory: *initiates(send-invoice(C, Item), payment-due(C, Item))*
terminates(pays-invoice(C, Item), payment-due(C, Item))

Initial state at sunday: *reliable(bob)*

External events: *orders(bob, book, monday), orders(mary, book, monday)*

A reactive model: *orders(bob, book, monday), orders(mary, book, monday)*
reliable(bob, sunday), reliable(bob, monday), reliable(bob, tuesday), etc.
send-invoice(bob, book, tuesday), dispatch(bob, book, tuesday),
payment-due(bob, book, tuesday), payment-due(bob, book, wednesday), etc.
sunday < monday, monday < tuesday, tuesday < wednesday, etc.

A proactive model: The reactive model with the addition of:
send-invoice(mary, book, tuesday), dispatch(mary, book, tuesday),
payment-due(mary, book, tuesday), payment-due(mary, book, wednesday), etc.

An irrelevant model: The reactive model with the addition of:
send-voucher(mary, wednesday).

Here the models are all Herbrand models, represented by the set of *all* atomic sentences that are *true* in the model. In particular, the models contain all the atomic sentences needed to define the temporal relations.

Fig. 1. Examples of Models of Reactive Rules in KELPS.

2 The KELPS Language

The operational semantics of KELPS maintains a single current state S_i at time i. It reasons with the reactive rules, to generate a set of actions $acts_{i+1}$, which it combines with external events ext_{i+1}, to produce a consistent set of concurrent events $ev_{i+1} = ext_{i+1} \cup acts_{i+1}$. The events ev_{i+1} are used to update the current state S_i, generating the successor state $S_{i+1} = succ(S_i, ev_{i+1})$.

In KELPS, states are represented by sets of ground atoms[2], also called *facts* or *fluents*. Events are also represented by ground atoms. Such sets of ground atoms can be understood in two ways: They can be understood literally as theories or as Herbrand interpretations, which are model-theoretic structures. It is this second interpretation that underpins the logical semantics of KELPS.

States and events can be represented with or without timestamps. The representation without timestamps facilitates destructive updates, because if a fact is not terminated by a set of events, then the fact without timestamps simply persists from one state to the next. However, the representation with timestamps makes it possible to combine all the states and events into a single Herbrand interpretation.

2.1 Vocabulary

KELPS is a first-order, sorted language, including a special sort for time. In the version of KELPS presented in this paper, we assume that time is linear and discrete, and that the succession of time points is represented by the ticks of a logical clock, where 1, 2, ... stand for *s(0), s(s(0)), ..., t+1* stands for *s*(*t*) and *t+n* stands for $s^n(t)$. Thus S_i represents the state at time i, and ev_{i+1} represents the set of events taking place in the transition from state S_i to S_{i+1}. Other representations of time are also possible, as illustrated informally in Fig. 1.

Predicates: The predicate symbols of the language are partitioned into sets representing fluents, events, auxiliary predicates and meta-predicates:

Fluent predicates represent facts in the states S_i. The last argument i of a timestamped fluent atom $p(t_1, ..., t_n, i)$ is a time parameter, representing the time i of the state S_i to which the fluent belongs. The unstamped fluent atom $p(t_1, ..., t_n)$ is the same atom without this timestamp. Fluents can also have other time parameters, called *reference times*, for example *friday* in *payment-due(friday, i)*, which expresses that at time i payment is due on *friday*.

Event predicates represent events contributing to the transition from one state to the next. The last argument of a timestamped event atom $e(t_1, ..., t_n, i)$ is a time parameter, representing the time of the successor state S_i. The unstamped event atom $e(t_1, ..., t_n)$ is the same atom without this time parameter. Event predicates are partitioned into *external event predicates* and *action predicates*.

Auxiliary predicates are of two kinds: *time-independent predicates*, such as *isa(book, product)*, and *temporal constraint predicates*, which express *temporal constraints*, including inequalities of the form $i < j$ or $i \leq j$ between time points, other

[2] By *atom* we mean an atomic formula possibly containing variables. By *atomic sentence* or *ground atom*, we mean an atomic formula not containing variables.

relationships between time points, such as *max(T1, T2, T)* and *min(T1, T2, T)*, and arithmetic relationships involving time points, such as *plus(T1, 3, T2)*.

In LPS, auxiliary predicates are defined by logic programs. In KELPS, they are defined more simply by a (possibly infinite) set ***Aux*** of atomic sentences. This assumption that the temporal constraint predicates are defined by sets of ground atoms is the same as the assumption made in the semantics of constraint predicates in constraint logic programming (CLP) [6]. The KELPS OS exploits this relationship with CLP by using a constraint solver to check temporal constraints for satisfiability. As is common in the theory of CLP, it is sufficient to ensure that constraints are satisfiable. However, in practice, it is useful also to simplify the constraints.

The *meta-predicates* consist of the two predictates *initiates(events, fluent)* and *terminates(events, fluent)*, which are used to specify the postconditions of events and to perform state transitions, as illustrated in Fig. 1. The first argument is a set of events, to cater for the case where two events together have different effects from the individual events on their own (such as buying two books and getting the cheaper one for half price). In LPS, these meta-predicates are defined by a logic program. In KELPS, they are defined by a set of atomic sentences [3] in a *causal theory* ***C***, which also contains constraints on the preconditions and co-occurrence of events.[4]

2.2 KELPS Framework

Definition. A KELPS framework (or program) is a triple <***R***, ***Aux***, ***C***>, where ***R*** is a set of reactive rules, ***Aux*** is a set of ground atoms defining auxiliary predicates, and ***C*** is a causal theory.

Rules in ***R*** are constructed from formulas that represent complex events or plans, expressed as conjunctions of state conditions, event atoms, and temporal constraints. Operationally, state conditions are queries to the current state, treated as a database. Like relational database queries, state conditions can be formulas of FOL. For example, the state condition $\forall It\ \forall D$ [*manages(M, D, T)* $\wedge$ *item(It, D)* $\rightarrow$ *instock(It, T)*] behaves as a query that returns managers *M* all of whose departments *D* have all of their items *It* in stock at time *T*.

Definition. A *state condition* is an FOL formula whose atoms are either time-independent predicates or fluent atoms having the same timestamp, which is unbound in the condition.

Rules can have disjunctive consequents. For example:

orders(C, Item, T1)
$\rightarrow$ [*dispatch(C, Item, T2)* $\wedge$ *send-invoice(C, Item, T3)* $\wedge$ *T1 < T2* $\leq$ *T3* $\leq$ *T1 + 3*]
$\vee$ [*send-apology(C, Item, T4)* $\wedge$ *T1 < T4* $\leq$ *T1 + 5*]]

[3] In the examples, in Fig. 1 and elsewhere in the paper, we use universally quantified sentences as a shorthand for the set of all their well-sorted ground instances.

[4] In earlier papers, this causal theory was called a "domain theory".

Different alternatives in the consequent can have different deadlines. If the antecedent becomes true, then the plan with the earliest deadline can be attempted first. If it fails to be achieved within the deadline, then an alternative plan with a later deadline can be attempted. However, any actions performed in the earlier, partially executed plan cannot be directly undone. Any "backtracking" to try an alternative plan must be performed in the context of the state updated by the successful actions in the failed plan.

Definition. A *reactive rule* is a sentence of the form:

$\forall X\ [antecedent \rightarrow \exists Y\ [consequent]]$ where:

- *consequent* is a disjunction $consequent_1 \vee ... \vee consequent_n$.
- X is the set of all variables, including time variables, that occur in *antecedent* and are not bound in state conditions. Y is the set of all variables, including time variables, that occur only in *consequent* and are not bound in state conditions.
- *antecedent* and each $consequent_i$ is a conjunction of state conditions, event atoms and temporal constraints.
- The only variables occurring in temporal constraints are those that occur in state conditions and event atoms of the rule, or ones that are functionally dependent on such variables.
- All the timestamps in Y are constrained in the *consequent* to be later than or equal to the timestamps in X.[5]

Because of the restrictions on the quantification of variables, and the logical equivalence $\exists Y[p \vee q] \Leftrightarrow \exists Y\ p \vee \exists Z\ q$, we can omit the quantifiers $\forall X$ and $\exists Y$, and simply write $antecedent \rightarrow consequent$ or $antecedent \rightarrow consequent_1 \vee ... \vee consequent_n$.

Definition. A *causal theory*, $\boldsymbol{C} = C_{post} \cup C_{pre}$, consists of two parts: C_{post} is a set of atomic sentences defining the predicates *initiates* and *terminates*. C_{pre} is a set of sentences of the form $\forall X\ [antecedent \rightarrow false]$, where *antecedent* is a conjunction consisting of a single state condition with timestamp T and event atoms with timestamp $T+1$, where T is included in X.

For example, the preconditions for *dispatch(C, Item, T)* in Fig. 1 might include:

$$dispatch(C1, Item, T) \wedge dispatch(C2, Item, T) \wedge C1 \neq C2 \rightarrow false$$
$$dispatch(C, Item, T+1) \wedge \neg\ instock(Item, T) \rightarrow false$$

where *instock(Item)* is a fluent, initiated and terminated by the actions *stock(Item*) and *dispatch(C, Item)*, respectively.

The definitions of the predicates *initiates* and *terminates* by means of atomic sentences is similar to the use of add-lists and delete-lists in STRIPS [19]. However, it is more general, because the first argument is a *set* of events. Stating explicitly the fluents initiated and terminated by every possible set of concurrent events is not very

[5] More precisely, for every substitution σ that replaces the time variables in X and Y by ground times and such that the temporal constraints in *consequent* σ are *true* in $\boldsymbol{Aux}$, the all timestamps in *consequent* σ are later than or equal to the latest timestamp in *antecedent* σ.

practical, but it clarifies the model-theoretic semantics and simplifies the operational semantics. Moreover, it paves the way for the more practical representation in which the *initiates* and *terminates* predicates are defined by logic programs in LPS.

Reactive rules $\boldsymbol{R}$ and preconditions C_{pre} have the same semantics. Moreover, they both have a deontic (but non-modal) character. The reactive rules specify obligations that must be fulfilled, typically by performing actions, whereas the preconditions specify combinations of state conditions and actions that are prohibited.

3 The KELPS Model-theoretic Semantics

In the model-theoretic semantics of KELPS, the reactive rules $\boldsymbol{R}$ and preconditions C_{pre} are interpreted according to the standard, non-modal semantics of classical first-order logic. This contrasts with the semantics of modal logics, in which states are represented by possible worlds, linked by accessibility relations. In KELPS, states and events are timestamped and included in a single model-theoretic structure.

Definition. If <$\boldsymbol{R}$, $\boldsymbol{Aux}$, $\boldsymbol{C}$> is a KELPS framework, S is a set of unstamped fluents, representing a single state, and ev is a set of unstamped events, representing concurrent events, then the associated *successor state* is:

$succ(S, ev) = (S - \{p \mid terminates(ev, p) \in C_{post}\}) \cup \{p \mid initiates(ev, p) \in C_{post}\}$.

Notation. If S_i is a set of fluents without timestamps, then S_i* represents the same set of fluents with the timestamp i. If $events_i$ is a set of concurrent events without timestamps, all taking place in the transition from state S_{i-1} to state S_i, then $events_i$* represents the same set of events with the timestamp i.

If S_0 is an initial state, $ext_1, \ldots, ext_i, \ldots$, is a sequence of sets of external events and $acts_1, \ldots, acts_i, \ldots$ is a sequence of sets of actions, then:

$\boldsymbol{S^*} = S_0^* \cup S_1^* \cup \ldots \cup S_i^* \cup \ldots$ where $S_{i+1} = succ(S_i, ev_{i+1})$

$\boldsymbol{ev^*} = ev_1^* \cup ev_2^* \cup \ldots \cup ev_i^* \cup \ldots$ where $ev_i = ext_i \cup acts_i$, for $i \geq 1$.

Computation in conventional reactive systems consists in generating a stream $act_1, \ldots, act_i, \ldots$ of actions in response to a stream of external events $ext_1, \ldots ext_i, \ldots$. Computation in KELPS is similar, but it has a purpose, which conventional reactive systems lack, namely to make rules and the preconditions of actions *true*:

Definition. Given a KELPS framework <$\boldsymbol{R}$, $\boldsymbol{Aux}$, $\boldsymbol{C}$>, an initial state S_0 and a sequence $ext_1, \ldots, ext_i$ of sets of external events, the *computational task* is to generate sets $acts_{i+1}$ of actions for every $i \geq 0$, such that $\boldsymbol{R} \cup C_{pre}$ is *true* in the Herbrand interpretation $\boldsymbol{M} = \boldsymbol{Aux} \cup \boldsymbol{S^*} \cup \boldsymbol{ev^*}$.

The definition of truth is the classical definition for FOL. It allows the generation of actions that make the rules *true* by making their antecedents *false*, or by making their consequents *true* whether their antecedents are *true* or *false*. It also allows actions that are irrelevant to the task. In this paper, we identify the reactive models that can be generated by the OS.

Note that the generated actions $acts_{i+1}$ in KELPS need not be a direct reaction to the current situation $S_i^* \cup ev_i^*$, but can be a partial response to some subsequence of earlier states and events. Nor need the choice of actions be deterministic. There can be several different sets of actions $acts_{i+1}$ that can be chosen for execution in the transition from a given state S_i to the next state S_{i+1} and there can be many different models $\boldsymbol{M}$ that satisfy a given computational task. However, once an action has been chosen and successfully executed, it cannot be directly undone. At best, it may be possible only to choose and execute other actions to reverse its effects.

3.1 Herbrand interpretations

The semantics of Herbrand interpretations is a simplified version of the standard semantics of first-order logic.

Definition. Given the vocabulary of a sorted first-order language, the *Herbrand universe* is the set of all well-sorted *ground* (i.e variable-free) terms that can be constructed from the constants and function symbols of the vocabulary. The *Herbrand base* is the set of all well-sorted ground atoms that can be constructed from the predicate symbols and the ground terms of the vocabulary. A *Herbrand interpretation* is a subset of the Herbrand base. A *Herbrand model M* of a set *S* of sentences is a Herbrand interpretation such that every sentence *s* in *S* is *true* in *M*.

The main difference from the standard definition of truth is the base case: If I is a Herbrand interpretation, then a ground atom A is *true* in I if and only if $A \in I$. Thus, a rule $\forall X\ [antecedent \rightarrow \exists Y\ [consequent_1 \vee ... \vee consequent_n]]$ is *true* in I if and only if, for every ground instance $antecedent\ \sigma$ that is *true* in I, there exists a ground instance $consequent_i\ \sigma\ \theta$ that is also *true* in I. Here the substitutions σ and θ replace the variables X and Y, respectively, by terms of the appropriate sort in the Herbrand universe U of I. For simplicity, we assume that, except for time parameters, all fluents have the same ground instances over U in all states.

3.2 The temporal structure of KELPS interpretations

The timestamping of fluents and events, and the restrictions on the syntax of KELPS provide KELPS interpretations with a rich structure of sub-interpretations. This structure is captured by the following theorem, which is an immediate consequence of the definition of truth for sentences in Herbrand interpretations.

Theorem 1.
1. If *s* is a conjunction of temporal constraints whose time parameters are all ground, then *s* is *true* in $\boldsymbol{Aux} \cup \boldsymbol{S^*} \cup \boldsymbol{ev^*}$ if and only if *s* is *true* in $\boldsymbol{Aux}$.
2. If *s* is an FOL sentence containing only state conditions, event atoms and temporal constraints whose time parameters are all ground, then:
   a. If all the timestamps in *s* are the same time *i*,
      then *s* is *true* in $\boldsymbol{Aux} \cup \boldsymbol{S^*} \cup \boldsymbol{ev^*}$ if and only if *s* is *true* in $\boldsymbol{Aux} \cup S_i^* \cup ev_i^*$.
   b. If *i* is the latest timestamp in *s*, then *s* is *true* in $\boldsymbol{Aux} \cup \boldsymbol{S^*} \cup \boldsymbol{ev^*}$

if and only if *s* is *true* in $\boldsymbol{Aux} \cup S_0^* \cup ... \ S_i^* \ \cup ev_1^* \cup ... \ ev_i^*$.

There is an obvious similarity with the possible world semantics of modal logic. Each sub-interpretation $\boldsymbol{Aux} \cup S_i^* \cup ev_i^*$ is analogous to a possible world, and the single interpretation $\boldsymbol{Aux} \cup \boldsymbol{S^*} \cup \boldsymbol{ev^*}$ is analogous to a complete frame of possible worlds and accessibility relations. However, in KELPS, timestamped fluents and events are all contained in a single Herbrand interpretation; but in the possible world semantics, fluents belong to possible worlds, and events belong to accessibility relations.

3.3 Sequential notation

The antecedents and alternative consequents of reactive rules are both partially ordered state conditions and event atoms. Antecedents represent complex (or composite) events, and alternative consequents represent conditional plans of actions. Although these state conditions and event atoms are partially ordered, they are used to recognize or generate linearly ordered sequences of states and events.

It is useful to have a notation that distinguishes between the different sequences represented by the same conjunction of partially ordered state conditions and event atoms:

Notation. Let *condition1* $\wedge$ *condition2* be a conjunction of state conditions and event atoms, *constraints* a conjunction of temporal constraints, and *C* the conjunction *condition1* $\wedge$ *condition2* $\wedge$ *constraints.* If there exists a substitution σ that grounds all the time parameters of *C*, and if *constraints* σ is true in ***Aux***, then:

1. *condition1* $<$ *condition2* $\wedge$ *constraints* denotes that for every timestamp t_1 in *condition1* σ and every timestamp t_2 in *condition2* σ , $t_1 < t_2$.
2. *condition1* $\leq$ *condition2* $\wedge$ *constraints* denotes that for every timestamp t_1 in *condition1* σ and every timestamp t_2 in *condition2* σ , $t_1 \leq t_2$.

We refer both to *condition1* $\leq$ *condition2* $\wedge$ *constraints* and to *condition1* $\leq$ *condition2* $\wedge$ *constraints* as *sequencings* of *C*. Note that *condition1* $<$ *condition2* and *condition1* $\leq$ *condition2* hold when *condition1* or *condition2* are empty (equivalent to *true*).

3.4. Reactive interpretations

Fig. 1 gives examples of different kinds of models of a KELPS program. The following definition characterizes reactive interpretations, which include reactive models as a special case. Loosely speaking, an action occurs in a reactive interpretation if and only if it occurs in the consequent of an instance of a reactive rule, and all earlier state conditions and event atoms in the instance are already *true*.

Definition. Given a KELPS framework <***R***, ***Aux***, ***C***>, initial state S_0 and set ***ev**** of timestamped events, let C_{pre} be *true* in $\boldsymbol{I} = \boldsymbol{Aux} \cup \ \boldsymbol{S^*} \cup \boldsymbol{ev^*}$, and let $\boldsymbol{ev^*} = \boldsymbol{ext^*} \cup \boldsymbol{acts^*}$ be a partitioning of ***ev**** into external events ***ext**** and actions ***acts****. Then ***I*** is *reactive* if and only if, for every action *act* in ***I***, there exists a rule $r \in \boldsymbol{R}$ of the form:

antecedent $\rightarrow$ [*other* $\vee$ [*earlier* $\wedge$ *action* $\wedge$ *remainder* $\wedge$ *temp*]] where:

1) *temp* consists of all the temporal constraints in *earlier*∧*action*∧*remainder*∧ *temp*
2) all the non-timestamp variables in *action* occur in *antecedent* or *earlier*[6] and
3) there exists an instance $r\ \sigma$ of r such that:
   a) *act* is *action* σ
   b) *antecedent* σ ∧ *earlier* σ ∧ *action* σ ∧ *temp* σ is *true* in ***I*** and
   c) *earlier* σ < *action* σ ≤ *remainder* σ ∧ *temp* σ.

I is a *reactive model* of <***R***, ***Aux***, ***C***>, if and only if ***R*** is *true* in ***I.***

4. The KELPS Operational Semantics

The operational semantics exploits the internal structure of KELPS interpretations ***Aux*** ∪ ***S**** ∪ ***ev**** to generate them by *progressively* extending a partial interpretation ***Aux*** ∪ S_0* ∪ ... S_i* ∪ ev_1* ∪ ... ev_i* one step at a time. Moreover, it does so by maintaining only the current state S_i and the events ev_i that gave rise to S_i, without remembering earlier states and events. For this purpose, it maintains a current set of partially evaluated rules R_i, which need to be monitored in the future, and a current goal state G_i, which needs to be made *true* in the future.

To deal with complex events in the *antecedents* of rules without remembering past states and events, the OS maintains a current set of rules R_i, starting with R_0 = ***R***. Each rule in R_i is the instantiated remainder *later* σ ∧ *temp* σ → *consequent* σ of a rule *earlier* ∧ *later* ∧ *temp* → *consequent* in ***R*** whose earlier part *earlier* σ is already *true*.

Logically, the goal state G_i is a conjunction of disjunctions. Each disjunct of a disjunction is the instantiated remainder *later* σ ∧ *temp* σ of a rule *antecedent* → [*other* ∨ [*earlier* ∧ *later* ∧ *temp*]] in ***R*** whose earlier part *antecedent* σ ∧ *earlier* σ is already *true*. Because of their similarity to goal clauses in logic programming, such disjuncts are also called *goal clauses* in KELPS.

Operationally, the goal state is a set (conjunction) of independent threads, and each thread is a goal tree, whose non-root nodes are goal clauses. The goal tree representation helps to structure the search space of alternative plans, and to guide the search strategy for trying different alternatives. If the goal trees are searched in a depth-first fashion, then they can be implemented by stacks, as in Prolog. Backtracking is possible, but previously generated actions and states cannot be undone.

The following specification of the OS is very abstract and ignores many optimizations that can improve efficiency. These are described in earlier papers [12, 13, 14, 15]. Some of these optimizations restrict the models that can be generated, and hence affect the relationship between the interpretations generated by the OS and the interpretations sanctioned by the model-theoretic semantics.

In the following definition, the OS is presented as an agent cycle. At the end of the cycle, external events are input and combined with selected actions. The resulting combined set of events is used to update the current state. In other versions of the OS, these updates were performed at the beginning of the cycle.

---

[6] This is a form of range restriction, which ensures that when an action is selected for execution, all its non-timestamped variables are grounded.

Definition. The OS Cycle. Initially S_0 is given, $R_0 = \boldsymbol{R}$, $G_0 = \{\}$ and $ev_0 = \{\}$.
For $i \geq 0$, given S_i, R_i, G_i, and ev_i, the *i-th* cycle consists of the following steps:

Step 1. Evaluate antecedents of rules. For every sequencing *current* θ < *rest* $\theta \wedge$ *constraints* θ of the antecedent of an instance $r\theta$ of a rule r

$$current \wedge rest \wedge constraints \rightarrow consequent$$

in R_i, where *current* θ is *true* in $\boldsymbol{Aux} \cup S_i^* \cup ev_i^*$ and θ instantiates only the variables in *current* and any evaluable time variables in *constraints* that are functionally dependent on the timestamp of *current*, add *rest* $\theta \wedge$ *constraints* $\theta \rightarrow$ *consequent* θ as a new reactive rule to R_i.
If *rest* θ is empty (equivalent to *true*) and *constraints* θ is *true* in $\boldsymbol{Aux}$ then transfer *consequent* θ from R_i to G_i, starting a new thread, which is a goal tree with *consequent* θ at the root. Add each disjunct of *consequent* θ whose constraints are satisfiable as a child of the root.

Step 2. Evaluate state conditions and simple event atoms in goal clauses. Choose a set of sequencings:

$$current\ \theta < rest\ \theta \wedge constraints\ \theta$$

of instances $C\theta$ of goal clauses C from one or more threads in G_i, where *current* θ is *true* in $\boldsymbol{Aux} \cup S_i^* \cup ev_i^*$ and θ instantiates only the variables in *current* and any evaluable time variables in *constraints* that are functionally dependent on the timestamp of *current*. For each such choice, add *rest* $\theta \wedge$ *constraints* θ to G_i, as a child of C.

Step 3. Choose a conjunction of actions for attempted execution. Choose a set of sequencings:

$$actions\ \tau \preceq rest\ \tau \wedge constraints\ \tau$$

of instances $C\tau$ of goal clauses C from one or more threads in G_i, where τ instantiates only the timestamp variables in *actions*, and *actions* τ is the conjunction of *all* the ground action atoms in $C\tau$ that have the same timestamp $i+1$. Let $candidate\text{-}acts_{i+1}$ be the set of all the action atoms in all such *actions* τ.

Step 4. Update S_i, G_i, R_i. Choose a subset $acts_{i+1}^* \subseteq candidate\text{-}acts_{i+1}$ such that C_{pre} is *true* in $\boldsymbol{Aux} \cup S_i^* \cup ev_{i+1}^*$, where $ev_{i+1}^* = ext_{i+1}^* \cup acts_{i+1}^*$ and the set of external events ext_{i+1}^* is given. Let $S_{i+1} = succ(S_i, ev_{i+1})$. Let $G_{i+1} = G_i$ and $R_{i+1} = R_i$.

Note that the OS allows attempting to make an instance of a consequent of a reactive rule true even though the same instance of the consequent has already been made true. This can be avoided easily in the OS, but would make the corresponding definition of reactive interpretations substantially more complex. However, there are other optimi-

sations that can also be made easily in the OS, without affecting the definition of reactive interpretation. These optimisations include removing from R_i rules whose antecedents are timed out, and removing from G_i leaf node goal clauses containing a conjunct that is timed out.

5 Relationships between the Model-theoretic Semantics and the Operational Semantics

The proof of soundness for the OS of LPS [11, 12, 14] also applies to KELPS:

Theorem 2. Soundness. Given a KELPS framework <***R***, ***Aux***, ***C***>, initial state S_0 and sequence $ext_1,\ldots, ext_i,\ldots$ of sets of external events, suppose that the OS generates the sequences $acts_1*,\ldots, acts_i*,\ldots$ of actions and $S_1*,\ldots, S_i*,\ldots$ of states. Then $\boldsymbol{R} \cup C_{pre}$ is true in ***I* = *Aux*** $\cup$ ***S**** $\cup$ ***ev**** if, for every goal tree that is added to a goal state G_i, $i \geq 0$, the goal clause *true* is added to the same goal tree in some goal state G_j, $j \geq i$.

The following theorems characterise the interpretations generated by the OS.

Theorem 3. The OS generates only reactive interpretations. Given a KELPS framework <***R***, ***Aux***, ***C***>, initial state S_0 and set of external events ***ext****, let ***acts**** be the set of actions generated by the OS, and ***ev**** = ***ext**** $\cup$ ***acts****. Then ***I* = *Aux*** $\cup$ ***S**** $\cup$ ***ev**** is a reactive interpretation.

Theorem 4. The OS can generate any reactive interpretation. Given a KELPS framework <***R***, ***Aux***, ***C***>, initial state S_0 and set of external events ***ext*,** let ***acts**** be a set of actions such that ***I* = *Aux*** $\cup$ ***S**** $\cup$ ***ev**** is a reactive interpretation. Then there exist choices in steps 2, 3 and 4 such that the OS generates ***acts**** (and therefore generates ***I***).

6 Related Work

In terms of expressive power, KELPS is similar to Reaction RuleML [16], and much of the comparison with other systems presented in [16] also holds for KELPS. Moreover, our earlier papers [11, 12, 13, 14] also include extensive discussions of the relationships between LPS and production systems, BDI agents, event-condition action rules in active databases, action languages in AI and other models of computation. For lack of space and to avoid repeating these comparisons, in this paper we will focus instead on pointing out only the most important relationships, which are with abductive logic programming, MetateM and Transaction Logic.

6.1 Abductive Logic Programming (ALP)

Despite the fact that logic programming plays only a supporting role in LPS, and no role at all in KELPS, ALP played an important role in the development of LPS, and therefore of KELPS. More importantly, the origin [9, 10] of LPS [11, 12, 14] in ALP

[8] helps to explain the semantics of KELPS and the issues concerning completeness, which are the focus of this paper.

An ALP framework is a triple <***L***, ***I***, ***A***>, where ***L*** is a logic program, ***I*** is a set of integrity constraints, and ***A*** is a set of atomic sentences, which are candidate assumptions. KELPS is closely related to the special case of ALP in which ***L*** is a set of atomic sentences including S_0, ***ext**** and ***Aux***, ***I*** consists of reactive rules ***R*** and preconditions C_{pre}, and ***A*** is the set of all possible actions. The biggest difference is the way in which KELPS generates ***S****.

In applications of ALP to planning problems [3, 18], it has been common to include the event calculus [15] in the logic program ***L***. Although this use of the event calculus has been interpreted as a solution to the frame problem [17], we believe that it cannot compete for efficiency with destructive change of state. However, destructive change of state does not have an obvious logical semantics. In particular, if states are regarded as syntactic objects, defined by axioms, then it is not possible to change the axioms during the course of trying to prove a theorem.

In KELPS, we solve this problem by regarding states and events as belonging to model-theoretic structures. This corresponds to a semantics for ALP in which, given an ALP framework <***L***, ***I***, ***A***>, an *abductive solution* is a subset Δ of ***A***, such that ***I*** is *true* in a canonical model of $\mathbf{L} \cup \Delta$. This is very close to the semantics of KELPS.

The issues concerning the completeness of KELPS are of two different kinds. The first is inherited from the semantics of abduction, namely that the semantics allows models in which Δ contains irrelevant actions. The second results from replacing the event calculus by destructive updates.

In the case of abduction, the first issue is dealt with by imposing further restrictions on the solutions Δ - for example, requiring that Δ be minimal, in the sense that no $\Delta' \subset \Delta$ (properly contained in Δ) is also a solution. But guaranteeing minimality is computationally expensive, and in practice some weaker, often informally specified requirement, such as relevance, is imposed. In the case of KELPS, the analogous relevance requirement is that generated actions be instances of action atoms that occur explicitly in the consequent of a reactive rule.

The second issue does not arise in ALP when the event calculus is included in the program ***L***, because then the event calculus can be used to make facts *true* by generating events that initiate them, and to make facts *false* by generating events that terminate them. In the case of KELPS, the more restricted causal theory C_{post} is used only to update states with given sets of events.

6.2 MetateM

MetateM [1] is a temporal modal logic language in which a program consists of sentences of the logical form:

'past and present formula' **implies** 'present or future formula'

Computation consists in generating a model in which all such sentences are *true*. These programs are similar in spirit to the reactive rules of KELP and have a similar model-theoretic semantics. The main differences are that, in KELPS, time is repre-

sented explicitly, models are classical rather than modal, and models are constructed by means of destructive updates

Completeness has been shown [1] for propositional MetateM, without external events, maintaining the entire history of past states, backtracking from the future into the past in the search for a model, and encoding frame axioms in the reactive rules.

6.3 Transaction Logic

Transaction Logic [2] is a declarative, logic-based language for defining complex transactions, which update states of a logic program or database. Transactions in Transaction Logic have a logical, model-theoretic semantics defined in terms of paths between states, generated by means of destructive updates. Although there is no direct analogue of reactive rules, they can be simulated by means of transactions.

KELPS shares with Transaction Logic the view of computation as generating sequences of destructively updated states that can be viewed as databases, in contrast with conventional programming languages, in which states are simply collections of variable-value assignments. KELPS also shares with Transaction Logic the view that transactions are sequences of sets of actions and database queries expressed in full FOL.[7] The main differences are that in KELPS, transactions are the consequents of reactive rules that are triggered when the antecedents become *true*, time is represented explicitly, and all states, actions and events are combined into one model-theoretic structure.

7 Conclusions and Future Work

This paper does not exhaust all of the theoretical issues concerning the semantics of KELPS and LPS. However, there are also important practical issues concerning knowledge representation and implementation that need further work. In particular, there are two extensions of KELPS and LPS that would significantly improve their expressive power. One is to allow the consequents of reactive rules to contain conditions that also have antecedent-consequent form. This can be implemented simply by allowing the remainder generated in step 2 of the OS to have the form of a reactive rule. The other extension is to allow reactive rules to contain more complex event conditions. This extension also does not affect the semantics, and can be implemented, for example, by storing a history of past events.

There are a number of implementations of LPS. Focusing on a single implementation and making it available for wider use are the main priority for future work.

Acknowledgements. Many thanks to Howard Boley for encouraging us with this work, and to the referees for their helpful comments on the paper.

[7] Transaction Logic also uses logic programs both to define intentional database predicates and to define sequences of state conditions and events. This capability is also available in LPS, but has been eliminated from KELPS for simplicity.

References

1. Barringer, H., Fisher, M., Gabbay, D., Owens, R., & Reynolds, M. (1996). *The imperative future: principles of executable temporal logic*. John Wiley & Sons, Inc.
2. Bonner, A., Kifer, M. 1993. Transaction logic programming. In Warren D. S., (ed.), *Logic Programming: Proc. of the 10th International Conf.*, 257-279.
3. Eshghi, K. 1988. Abductive Planning with Event Calculus. In *ICLP/SLP*, pp. 562-579.
4. Gurevich, Y. (1995). Evolving algebras 1993: Lipari guide. *Specification and validation methods*, 9-36.
5. Harel, D. 1987. Statecharts: A Visual Formalism for Complex Systems, *Sci. Comput. Programming* 8, 231-274.
6. Jaffar, J., & Lassez, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of programming languages*, ACM, 111-119.
7. Kakas, A. C., Kowalski, R., Toni, F. 1998. The Role of Logic Programming in Abduction, *Handbook of Logic in Artificial Intelligence and Programming 5*, Oxford University Press, 235-324.
8. Kakas, A. C., Mancarella, P., Sadri, F.,Stathis, K, and Toni, F. 2004. The KGP model of agency, In *Proc. ECAI-2004*.
9. Kowalski, R. and Sadri, F. 1999. From Logic Programming Towards Multi-agent Systems, *Annals of Mathematics and Artificial Intelligence*, Vol. 25, 391-419.
10. Kowalski, R. and Sadri, F. 2009. Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents. In *Proceedings of The Third International Conference on Web Reasoning and Rule Systems*, Chantilly, Virginia, USA.
11. Kowalski, R. and Sadri, F. 2010. An Agent Language with Destructive Assignment and Model-Theoretic Semantics, In Dix J., Leite J., Governatori G., Jamroga W. (eds.), *Proc. of the 11th International Workshop on Computational Logic in Multi-Agent Systems (CLIM*A), 200-218.
12. Kowalski, R. and Sadri, F. 2011. Abductive Logic Programming Agents with Destructive Databases, *Annals of Mathematics and Artificial Intelligence*, Vol. 62, No. 1, 129-158.
13. Kowalski, R. and Sadri, F. 2012b. A Logic-Based Framework for Reactive Systems, *Rules on the Web: Research and Applications, 2012* – RuleML 2012, Springer-Verlag. A. Bikakis and A. Giurca (Eds.), LNCS 7438, pp. 1–15.
14. Kowalski, R. and Sadri, F. 2014. Model-theoretic and operational semantics for Reactive Computing. To appear in *New Generation Computing*.
15. Kowalski, R., Sergot, M. 1986. A Logic-based Calculus of Events. In: *New Generation Computing*, Vol. 4, No.1, 67—95.
16. Paschke, A., Boley, H., Zhao, Z., Teymourian, K., & Athan, T. 2012. Reaction RuleML 1.0: standardized semantic reaction rules. In *Rules on the Web: Research and Applications*, Springer Berlin Heidelberg, pp. 100-119.
17. Shanahan, M. 1997. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT press.
18. Shanahan, M. 2000. An abductive event calculus planner. *The Journal of Logic Programming*, *44*(1), 207-240.
19. Fikes, R. E., & Nilsson, N. J. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3), 189-208.