A Logical Characterization of a Reactive System Language

Robert Kowalski and Fariba Sadri
Department of Computing
Imperial College London

Contents

Our Logic-based Reactive Framework *KELPS*

- Motivation
- Some features
- Operational semantics
- Model theoretic semantics
- Formal properties

KELPS

LPS = Logic-based Production System-like language

- Logic Programs
  - Intensional part of a deductive database
  - Complex events and transactions
  - Causal Theory
- Reactive Rules
- Destructively updated extensional database

KELPS (Kernel of LPS)

LPS but without the Logic Programs

Heritage of LPS/KELPS

- Abductive Logic Programming
- AI
- Attempt to exploit efficiency of conventional databases

Motivation

- Explore a logical basis for state transition systems and reactivity
- Important in many areas of computing:
  - condition-action rules in production systems
  - event-condition-action rules in active databases
  - transition rules in Abstract State Machines
  - Implicitly in Statecharts and BDI agents plans
  - Core of Reaction RuleML

- KELPS is a reactive state transition system.
- It has a simple operational semantics.
- We investigate the logical (declarative) semantics of KELPS.
- In particular we investigate a declarative semantics for *reactivity*.

KELPS Framework <*R*, *Aux*, *C*>

R: *(Reactive)* Rules

$\forall X$ *[antecedent* $\rightarrow$ $\exists Y$ *[consequent]]*

E.g. *orders(C, Item, T1)* $\wedge$ *reliable(C, T1)* $\rightarrow$
dispatch(C, Item, T2) $\wedge$
send-invoice(C, Item, T3) $\wedge$
$T1 < T2 \leq T3 \leq T1 + 3$

External event
actions
temporal constraints
state condition

KELPS Framework <*R*, *Aux*, *C*>

Aux: Auxiliary predicates defined by ground atoms.

- *Time-independent predicates*, e.g.
  isa(book, product).
- *Temporal constraint predicates*, e.g.
  $i < j$ or $i \leq j$ between time points.

KELPS Framework <*R*, *Aux*, *C*>

C : Causal Theory = $C_{pre} \cup C_{post}$

C_{pre} : (Integrity constraints)

Preconditions and executability of concurrent actions

$\forall X$ [*antecedent* $\rightarrow$ *false*]

E.g. *dispatch(Cust1, Item, T)* $\wedge$ *dispatch(Cust2, Item, T)* $\wedge$ *Cust1* $\neq$ *Cust2* $\rightarrow$ *false*

dispatch(Cust, Item, T+1) $\wedge \neg$ *instock(Item, T)* $\rightarrow$ *false*

C_{post} : *initiates* and *terminates* defined by (ground) atoms.

initiates(events, fluent) and *terminates(events, fluent)*.

E.g. (shorthand) *initiates([send-invoice(C, Item)], payment-due(C, Item))*
terminates([pays-invoice(C, Item)], payment-due(C, Item))

Rules can have disjunctive consequents

E.g.:

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)* $\wedge$

$T1 < T4 \leq T1 + 5$]

KELPS combines composite event recognition and composite transactions

heat-sensed(A, T_1) $\wedge$ smoke-sensed(A, T_2) $\wedge$
$|T_1 - T_2| \leq$ 60 sec $\wedge$ max(T_1, T_2, T) $\rightarrow$
activate-sprinkler(A, T_3) $\wedge$ $T < T_3 \leq T + 10$ sec $\wedge$
on-duty(SecGuide, T_4) $\wedge$ send(SecGuide, A, T_4) $\wedge$
$T_3 < T_4 \leq T_3 + 30$ sec
$\vee$
call(fire-department, A, T_5) $\wedge$
$T < T_5 \leq T + 120$ sec

Many different actions can generate models that make the rules true.

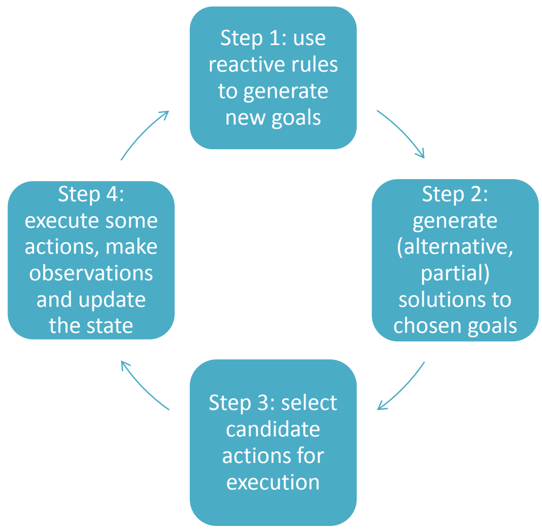
Time

Time is linear and discrete.

In the model-theoretic semantics
facts/fluents/tuples (in states) and
events are time-stamped and
included in a single model.

In the operational semantics
updates are performed destructively.

The Operational Semantics Cycle



S_0	$Goal\ State_0$	$R_0=R$
$\{ext_1 \cup acts_1\}$		
S_1	$Goal\ State_1$	R_1
…	…	
…	…	
S_i	$Goal\ State_i$	R_i
$\{ext_{i+1} \cup acts_{i+1}\}$		
S_{i+1}	$Goal\ State_{i+1}$	R_{i+1}
…	…	…

Database and Event Store

Database/State: Destructively updated by events via the Causal Theory.

Event store: Stores only the last events leading to the current database state. (events: external events and actions)

Note: Stream processing for CEP.

Goal State

Keeps track of the goals generated by the reactive rules, and of the alternative (partial) plans generated for solving them so far.

In general the consequent of a reactive rule is disjunctive.
So when a new goal is added to the goal state,
each disjunct of the goal whose constraints are satisfiable
is added as a child of the goal.

$$(c_1 \wedge \ldots \wedge c_m) \vee (d_1 \wedge \ldots \wedge d_n) \vee (e_1 \wedge \ldots \wedge e_p)$$

$c_1 \wedge \ldots \wedge c_m$ $d_1 \wedge \ldots \wedge d_n$ $e_1 \wedge \ldots \wedge e_p$

Fluents:

If p represents a fact/fluent in a state S_i

then $p(i)$ records the time i of S_i.

$S_i^* = \{p(i) \mid p \in S_i\}$

Events: Partitioned into *external events* and *actions*.

If e represents an event between S_i and S_{i+1}

then $e(i+1)$ records the time $i+1$ of S_{i+1}.

$ev_i^* = \{e(i) \mid e \in ev_i\}$

KELPS - Computing as Model Generation

Given $<R, Aux, C>$, S_0 and sets $ext_1, \ldots, ext_i$ of external events, the *computational task* is to generate sets $acts_{i+1}$ of actions, such that $R \cup C_{pre}$ is *true* in the Herbrand interpretation $M = Aux \cup S^* \cup ev^*$.

$S^* = S_0^* \cup S_1^* \cup \ldots \cup S_i^* \cup \ldots$ where

$S_{i+1} = (S_i - \{p \mid terminates(ev_{i+1}, p) \in C_{post}\}) \cup \{p \mid initiates(ev_{i+1}, p) \in C_{post}\}$.

$ev^* = ev_1^* \cup ev_2^* \cup \ldots \cup ev_i^* \cup \ldots$ where

$ev_i^* = ext_i^* \cup acts_i^*$.

Example of Models of a KELPS Program

R: *orders(C, Item, T1)* $\wedge$ *reliable(C, T1)* $\rightarrow$ *dispatch(C, Item, T2)* $\wedge$
send-invoice(C, Item, T3) $\wedge$ $T1 < T2 \leq T3 \leq T1 + 3$

Aux: *sun < mon, mon < tues, tues < wed, etc.*

C_{post}: *initiates([send-invoice(C, Item)], payment-due(C, Item))*
terminates([pays-invoice(C, Item)], payment-due(C, Item))

S_o: *reliable(bob)* on *sun.*

External events: *orders(bob, book1, mon),* *orders(mary, book2, mon)*

Reactive model: *Aux* $\cup$ *S** $\cup$ *ev** =

*ext**	*orders(bob, book1, mon),*		*orders(mary, book2, mon),*
*S**	*reliable(bob, sun),*	*reliable(bob, mon),*	*reliable(bob, tues), etc.*
*act**	*send-invoice(bob, book1, tues),*		*dispatch(bob, book1, tues),*
*S**	*payment-due(bob, book1, tues),*		*payment-due(bob, book1, wed), etc.*
Aux	*sun < mon,*	*mon < tues,*	*tues < wed, etc.*

The model-theoretic semantics allows non-reactive models

R: *orders(C, Item, T1)* $\wedge$ *reliable(C, T1)* $\rightarrow$
dispatch(C, Item, T2) $\wedge$ *send-invoice(C, Item, T3)* $\wedge$
$T1 < T2 \leq T3 \leq T1 + 3$

Proactive model:

The reactive model plus:

*act** *send-invoice(mary, book2, tues),* *dispatch(mary, book2, tues),*

*S** *payment-due(mary, book2, tues),*
payment-due(mary, book2, wed), etc.

Irrelevant model:

The reactive model plus:

*act** *send-voucher(mary, wed).*

Reactive Interpretations & Models

$\boldsymbol{I = Aux \cup S^* \cup ev^*}$ $\boldsymbol{ev^* = ext^* \cup acts^*}$ C_{pre} *true* in $\boldsymbol{I}$

For every $act \in acts^*$, there exists $r \in R$ and some σ such that

r is $antecedent \rightarrow [other \vee [earlier \wedge action \wedge remainder \wedge temp]]$

$r\sigma$ is $antecedent\ \sigma \rightarrow [other\ \sigma \vee [earlier\ \sigma \wedge act \wedge remainder\ \sigma \wedge temp\ \sigma]]$

true before *i*

true at *i*

could be true at or after *i*

true in *Aux*

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

The KELPS Operational Semantics (OS) is Sound

Given <*R*, *Aux*, *C*>, initial state S_0 and external events *ext**:

Theorem. If the OS generates *acts**, and
every goal *G* added to a goal state G_i
is reduced to *true* in some G_j, $j \geq i$,

then $R \cup C_{pre}$ is true in $I = Aux \cup S^* \cup ev^*$.

The KELPS OS
Generates only Reactive Interpretations

Given <*R, Aux, C*>, initial state S_0 and external events *ext**:

Theorem.

If the OS generates *acts** , and $ev^* = ext^* \cup acts^*$,

then $I = Aux \cup S^* \cup ev^*$ is a reactive interpretation.

The KELPS OS can
Generate any Reactive Interpretations

Given <*R, Aux, C*>, initial state S_0 and external events *ext**:

Theorem.

If $I = Aux \cup S^* \cup ev^*$ is a reactive interpretation,
where $ev^* = ext^* \cup acts^*$,
then there exist choices in *steps 2, 3 and 4* such that
the OS generates *acts** (and therefore generates *I*).

Conclusion and Further Work

- KELPS simplified kernel of LPS
- Suggested a characterisation of reactive interpretations and models
- Proved semantic properties of KELPS

Future Work:

- Explore semantic properties of full LPS
- Practical developments of LPS

Thank you.

Additional Slides

The remaining slides were not used in the main part of the talk but some were used when answering questions.

The OS Cycle: There are 4 steps
Step 1 (informally)

Reason forwards with the reactive rules: some may fire and set new goals.

E.g.

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

and event *orders(bob, book1) at time 1, with the database at time 1 containing* *reliable(bob)* will result in a new goal:

$dispatch(bob, book1, T2) \wedge send\text{-}invoice(bob, book1, T3) \wedge$
$1 < T2 \leq T3 \leq 4$

Step 1 (informally) cntd

Other reactive rules may be triggered, but may not fire yet. We keep the residue for future cycles.

E.g. Given
heat_sensor_detects(high_temperature, A, T_1) $\wedge$
smoke_detector_detects(smoke, A, T_2) $\wedge$ $T_2 - T_1 \leq 60$ *sec* $\rightarrow$
activate_sprinkler(A, T_3) $\wedge$ …

and event *heat_sensor_detects(high_temperature, area1) at time 5* *generates residue*

smoke_detector_detects(smoke, area1, T_2) $\wedge$ $T_2 - 5 \leq 60$ *sec* $\rightarrow$
activate_sprinkler(area1, T_3) $\wedge$ …

Helpful Notation: Sequences

- The antecedents and (alternative) consequents of reactive rules are partially ordered state conditions and event atoms.
- Although partially ordered, they are used to recognize or generate linearly ordered sequences of states and events.

Given *condition1* $\wedge$ *condition2* $\wedge$ *constraints* and substitution σ such that *constraints* σ is true.

Then *condition1* < *condition2* $\wedge$ *constraints* iff *t1* < *t2*

- for every time-stamp *t1* in *condition1* σ and
- for every time-stamp *t2* in *condition2* σ.

condition1 $\leq$ *condition2* $\wedge$ *constraints* iff *t1* $\leq$ *t2*

The OS Cycle: Step 1 (more formally)

Let S_0 be given, $R_0 = R$, $G_0 = \{\}$ and $ev_0 = \{\}$. Given S_i, R_i, G_i and ev_i:

Step 1. Evaluate antecedents of rules.

For every sequencing $current\ \theta < rest\ \theta \wedge constraints\ \theta$ of the antecedent of a rule *r*

$current < rest \wedge constraints \rightarrow consequent$ in R_i

such that $current\ \theta$ is true in $Aux \cup S_i^* \cup ev_i^*$

add $rest\ \theta \wedge constraints\ \theta \rightarrow consequent\ \theta$ to R_i.

If $rest\ \theta$ is empty, then transfer $consequent\ \theta$ to G_i.

The OS Cycle: Step 2

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, such that

$current\ \theta$ is true in $Aux \cup S_i^* \cup ev_i^*$

add $rest\ \theta \wedge constraints\ \theta$ to G_i as a child of *C*.

The OS Cycle: Steps 3 and 4

Step 3. Choose actions for attempted execution.

Choose *actions* $\leq$ *rest* $\wedge$ *constraints* in G_i

such that *actions* τ all have time $i+1$

Let $candidate\text{-}acts_{i+1}$ be the set of all such *actions* τ.

Step 4. Update the current state.

Choose $acts_{i+1}^* \subseteq candidate\text{-}acts_{i+1}$ such that

C_{pre} is true in $Aux \cup S_i^* \cup ext_{i+1}^* \cup acts_{i+1}^*$.

Update S_i to S_{i+1} . Let $G_{i+1} = G_i$ and $R_{i+1} = R_i$.