LPS and its Kernel KELPS: Logic-Based State Transition Frameworks

Fariba Sadri
Joint Work with Bob Kowalski
Imperial College London
Contents

1) Motivation
2) Language
3) Operational Semantics
4) Model Theoretic Semantics
5) Formal Properties
6) Examples
7) Conclusions
1. Motivation

- Explore a practical logical basis for state transition systems and reactivity
- Extend Logic Programming to include Reactive Programming and Destructive Updates, but with Logic-Based Semantics

Reactivity and State Transition are important in many areas of computing:

- condition-action rules in production systems
- event-condition-action rules, for example in active databases
- transition rules in Abstract State Machines
- Implicitly in Statecharts and BDI agents plans
2. Language

LPS – Broad Panorama

LPS = Logic-based Production System-like language

- Logic Programs
- Reactive Rules
LPS – Zooming In

LPS = Logic-based Production System-like language

- Logic Programs
  - Complex Event and Transaction Definitions
    - e.g. similar to Transaction Logic
    - Defined by a logic program
  - Database
    - Extensional (Destructively updated)
    - Intensional (Defined by a logic program)
  - Causal Theory
    - as in AI action theories, such as the event calculus (but without frame axioms)
    - Represented by a logic program

- Reactive Rules
KELPS (Kernel of LPS): LPS but without the Logic Programs, i.e. With only the boxed parts below:

- Logic Programs
  - Complex Event and Transaction Definitions
    - e.g. similar to Transaction Logic
    - Defined by a logic program
  - Database
    - Extensional (Destructively updated)
    - Intensional (Defined by a logic program)
- Causal Theory
  - as in AI action theories, such as the event calculus
  - Represented by a logic program
- Reactive Rules
Both LPS and KELPS have:

- Operational semantics
- Logical (declarative) semantics

We simplified LPS to KELPS primarily to facilitate more detailed theoretical analysis.

Despite its simplicity KELPS can still represent a variety of theories.
KELPS Framework \(<R, \text{Aux}, C>\)

\(R\): (Reactive) Rules

\[\forall X \ [\text{antecedent} \rightarrow \exists Y \ [\text{consequent}]]\]

- **consequent** is a disjunction
  \[\text{consequent}_1 \lor \ldots \lor \text{consequent}_n\]

- **antecedent** and each **consequent\(_i\)** are conjunctions of FOL conditions and temporal constraints.

- There are more details in the formal definition, for example about the time parameters – Please see the papers.
Examples of Reactive Rules $R$

**Shepherd:**

\[
\text{seeWolf}(\text{shep}, T) \rightarrow \text{cryWolf}(\text{shep}, T+1)
\]
\[
\text{cryWolf}(\text{shep}, T) \land \neg \text{help}(\text{shep}, T+1) \rightarrow \\
\text{cryWolf}(\text{shep}, T+2)
\]

**Villagers:**

\[
\text{cryWolf}(X, T) \land \neg \text{joker}(X, T) \rightarrow \text{help}(X, T+1)
\]
\[
\text{cryWolf}(X, T1) \land \neg \text{wolf}(T1) \land \text{cryWolf}(X, T2) \land \\
\neg \text{wolf}(T2) \land T1 < T2 \rightarrow \text{assume}(\text{joker}(X), T2+1)
\]

\[
\text{initiates}(\text{assume}(\text{joker}(X), \text{joker}(X)))
\]
Example of a more complicated Reactive Rule

\[ \text{orders}(C, \text{Item}, T1) \land \text{reliable}(C, T1) \rightarrow \]

\[ [\text{dispatch}(C, \text{Item}, T2) \land \text{send-invoice}(C, \text{Item}, T3) \land T1 < T2 \leq T3 \leq T1 + 3] \lor \]

\[ [\text{send-apology}(C, \text{Item}, T4) \land T1 < T4 \leq T1 + 5] ] \]
Reactive Rules can Represent

- Event-Condition-Action rules
- Event-Condition-Plan rules
- BDI-like plans
- Production rules
- Obligations
- Abstract State machines
KELPS Framework <R, Aux, C>

C : Causal Theory

\[ C = C_{pre} \cup C_{post} \]

\[ C_{pre} : \quad (\text{Integrity constraints}) \]
\[ \forall X [\text{antecedent} \rightarrow false] \]

➢ To constrain executability of concurrent actions

\[ \text{dispatch}(\text{Cust1, Item, T}) \land \text{dispatch}(\text{Cust2, Item, T}) \land \text{Cust1} \neq \text{Cust2} \rightarrow false \]

➢ To require co-existence of some actions:

\[ \text{leave\_house}(T) \land \neg \text{take\_keys}(T) \rightarrow false \]

➢ To specify preconditions of actions

\[ \text{give\_bonus}(M, T) \land \text{manager}(M, T) \land \exists D (\text{manages}(M, D, T) \land \text{loss\_making}(D, T)) \rightarrow false \]
$C_{post}$:  
initiates and terminates defined by (ground) atoms.  

\[
\text{initiates}(\text{events, fluent}) \quad \text{and} \quad \text{terminates}(\text{events, fluent}).
\]

E.g. (shorthand: Variables $C$ and $Item$ stand for ground instances)  

\[
\text{initiates}([\text{send\_invoice}(C, Item)], \text{payment\_due}(C, Item)) \\
\text{terminates}([\text{pays\_invoice}(C, Item)], \text{payment\_due}(C, Item))
\]
KELPS Framework \(<R, \text{Aux}, C>\)

**Aux**: Auxiliary predicates defined by ground atoms.

- *Time-independent predicates*, e.g.\(\text{isa(book, product)}\).
- *Temporal constraint predicates*, e.g.\(i < j\) or \(i \leq j\) between time points.
3. The Operational Semantics (OS): Cycle

Step 1: Use reactive rules to generate new goals

Step 2: Generate (alternative, partial) solutions to chosen goals

Step 3: Select candidate actions for execution

Step 4: Execute some actions, make observations and update the state
Step 1: Use reactive rules to generate new goals
Also to recognise complex events –

Stream Processing of Events

\[ \text{ev}_1(T_1) \land c_1(T_1) \land \text{ev}_2(T_2) \land c_2(T_2) \land \ldots \land \text{ev}_n(T_n) \land c_n(T_n) \land \text{constraints on } T_1, T_2, \ldots, T_n ] \rightarrow \text{consequent} \]
Step 2: Generate (alternative, partial) solutions to chosen goals

- Deliberative reasoning in LPS if we have clauses.

- Goal State is a forest of trees. The top level nodes of the trees are instances of the consequents of reactive rules that “have been fired”.

- The trees are extended deliberatively, each branch corresponding to one possible (partial) plan for solving the root goal.

\[(c_1 \land \ldots \land c_m) \lor (d_1 \land \ldots \land d_n) \lor (e_1 \land \ldots \land e_p)\]
Notes on the OS: Implementation

We have implemented different strategies for searching the space, e.g., based on:

✓ Priorities of reactive rules
✓ Deadlines given by the temporal constraints
✓ Length of time a goal has been waiting
✓ .....
Step 4: execute some actions, make observations and update the state

- Updating the state is destructive via the Causal Theory.
So we keep only the current state of the (database) state.

There is no Frame Axiom (common in AI causal theories). The frame axiom is an emergent property, not one to reason with in practice.

Event store: Stores only the latest events.
KELPS - Computing as Model Generation

\[ S_0 \]
\[ \{ext_1 \cup acts_1\} \]
\[ S_1 \]
\[ \ldots \]
\[ \ldots \]
\[ S_i \]
\[ \{ext_{i+1} \cup acts_{i+1}\} \]
\[ S_{i+1} \]

\[ Goal State_0 \]
\[ R_0 = R \]

\[ Goal State_1 \]
\[ R_1 \]

\[ \ldots \]
\[ \ldots \]

\[ Goal State_i \]
\[ R_i \]

\[ \ldots \]
\[ \ldots \]

\[ Goal State_{i+1} \]
\[ R_{i+1} \]

\[ \ldots \]
Contents

1) Motivation
2) Language
3) Operational Semantics
4) Model Theoretic Semantics
5) Formal Properties
6) Examples
7) Conclusions
4. Model Theoretic Semantics

KELPS - Computing as Model Generation

Given \( <R, \text{Aux}, C> \), \( S_0 \) and sets \( \text{ext}_1, \ldots, \text{ext}_i \) of external events, the \textit{computational task} is to generate sets \( \text{acts}_{i+1} \) of actions, such that \( R \cup C_{\text{pre}} \) is true in the Herbrand interpretation \( M = \text{Aux} \cup S^* \cup \text{ev}^* \).

\[
S^* = S_0^* \cup S_1^* \cup \ldots \cup S_i^* \cup \ldots \quad \text{where}
\]

\[
S_{i+1} = (S_i - \{ p | \text{terminates}(\text{ev}_{i+1}, p) \in C_{\text{post}} \}) \cup \{ p | \text{initiates}(\text{ev}_{i+1}, p) \in C_{\text{post}} \}.
\]

\[
\text{ev}^* = \text{ev}_1^* \cup \text{ev}_2^* \cup \ldots \cup \text{ev}_i^* \cup \ldots \quad \text{where}
\]

\[
\text{ev}_i^* = \text{ext}_i^* \cup \text{acts}_i^*.
\]
5. Formal Properties

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 $I = Aux \cup S^* \cup ev^*$.
What Interpretations/Models Does KELPS Generate?

Reactive rule:

\[ \text{seeWolf}(T) \land \text{outdoors}(T) \rightarrow \text{cryWolf}(T+1) \]

Initial State: outdoors
External event: seeWolf(3)
Causal Theory: terminates(goInside, outdoors)
initiates(goOutside, outdoors)

Reactive model: seeWolf(3), cryWolf(4)
Proactive model: cryWolf(1), cryWolf(2),
seeWolf(3), cryWolf(4)
Irrelevant model: seeWolf(3), cryWolf(4), drink(4)
Preventative model: outdoors(0), outdoors(1), goInside(1), seeWolf(3)

Formal definition of reactive models in our papers.
The KELPS OS
Generates only Reactive Interpretations

Given \(<R, \text{Aux, \(C\)}, \text{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 $I$).
The frame axiom is an emergent property

Given a (range restricted) KELPS framework $<R, Aux, C>$, initial state $S_0$ and sequence of sets of concurrent events $ev_0, ..., ev_i, ...$, where $ev_0 = \{\}$, let

$I = Aux \cup S^* \cup ev^*$, where

$S^* = S_0^* \cup ... \cup S_i^* \cup ...$ where

$S_{i+1} = succ(S_i, ev_{i+1})$ and

$ev^* = ev_0^* \cup ... \cup ev_i^* \cup ...$

Then for all time-stamped fluents $p(i)$ and for all $ev_i$:

$[initiates(ev_i, p) \rightarrow p(i)] \land$

$[p(i) \land \neg terminates(ev_i, p) \rightarrow p(i+1))]$

is true in $I$. 
Contents

1) Motivation
2) Language
3) Operational Semantics
4) Model Theoretic Semantics
5) Formal Properties
6) Examples
7) Conclusions
6. Examples of KELPS/LPS Formalisations

- BDI AgentSpeak Plans
- ECA Rules
- Abstract State machines - Conway Game of Life
- Obligations
BDI AgentSpeak Plans

e: \( b_1 \land \ldots \land b_m \leftarrow h_1; \ldots; h_n \)

- **event**
- **context**
- **goals/actions**

- e: a triggering event
- \( b_1, \ldots, b_m \): belief literals
- \( h_1, \ldots, h_n \): goals or actions
BDI Example

+location(waste, X) : location(robot, X) & location(bin, Y) ←
pick(waste);
! Location(robot, Y);
drop(waste).

Notice that a logical reading of this does not make sense, although the claim is that “This language ... allows agent programs to be written and interpreted in a manner similar to that of horn-clause logic programs”.

AgentSpeak(L): BDI Agents speak out in a logical computable language, Anand S. Rao
In LPS

\[ \text{location}(\text{waste, } X, T1) \land \text{location}(\text{robot, } X, T1) \land \text{location}(\text{bin, } Y, T1) \]

\[ \rightarrow \text{pick}(\text{waste, } T1+1) \land \text{goto}(\text{robot, } Y, T2) \land T2 > T1 \]

\[ \text{drop}(\text{waste, } T2+1) \]
goto(robot, Y, T) ← location(robot, Y, T)
goto(robot, Y, T2) ←

location(robot, X, T1) ∧ ¬X = Y ∧
adjacent(X, Z) ∧
¬location(car, Z, T1) ∧
move(robot, Z, T1) ∧
goto(robot, Y, T2) ∧ T1 < T2
ECA Rules

Hospital Example

duty_nurse(N, Ward, T) \land
spot_stranger(N, Ward, T) \rightarrow
  \text{stream\_videoCam}(N, Ward, T_1) \land
  \text{set\_off\_alarm}(N, Ward, T_2) \land
  T_1 < T+3 \land T_2 < T+3

\text{duty\_nurse}(N, Ward, T) \land
\text{emergency\_alert}(Patient, Ward, T) \rightarrow
  \text{duty\_head\_nurse}(HN, T) \land
  \text{inform}(N, HN, Patient, Ward, T+1) \land
  \text{take\_emergency\_kit}(N, Patient, Ward, T+2)
Abstract State Machines
Conway Game of Life

- Grid of square *cells*, each of which is in one of two possible states, *alive* or *dead*.
- At each step in time, the following transitions occur:
  - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
  - Any live cell with two or three live neighbours lives on to the next generation.
  - Any live cell with more than three live neighbours dies, as if by overcrowding.
  - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

- The initial pattern constitutes the *seed* of the system.
In LPS/KELPS

\[
\text{aliveNeighb}(C, N, T) \land (N < 2 \lor N > 3) \land \text{alive}(C, T) \rightarrow \\
\text{retract} (\text{alive}(C), T+1)
\]

\[
\text{aliveNeighb}(C, N, T) \land N = 3 \land \neg \text{alive}(C, T) \rightarrow \\
\text{assert} (\text{alive}(C), T+1)
\]

\text{aliveNeighb}/3 can be defined by LPS logic programming clauses, or replaced in the reactive rules with its definition.
Obligations
SBVR Example

SBVR: Semantics of Business Vocabulary and Business Rules

• It is obligatory that the supplier ensure to the purchaser that the service is replaced within 3 days from the notification if the service is not under quality of service agreement.

• It is obligatory that the supplier ensure to the purchaser that the service is refunded and a penalty of $1000 is paid if the service is not replaced within 3 days.
In KELPS

\[ \text{notify}(P, S, \text{Ser}, T1) \land \\
\neg \text{covered}_{\text{under}}(\text{Ser}, \text{quality}_{\text{of}}_{\text{service}}, T1) \rightarrow \\
[[\text{replace}(S, P, \text{Ser}, T2) \land T2 \leq T1+3] \lor \\
[\text{refund}(S, P, \text{Ser}, T3) \land \\
\text{pay}_{\text{penalty}}(S, P, \text{Ser}, $1000, T3) \land T3 > T1+3]] \]
Conclusions

LPS combines

- Reactive Rules,
- Causal Theories, and
- Logic Programs

in a single, practical framework with a logical model theoretic semantics.

This combination seems to lend itself well to represent state transitions.
We would welcome:

- Comments
- Collaboration on:
  - Research
  - PhD supervision
  - Implementation
  - Application development
Some Papers


Thank you for listening.

Questions