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

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*, *Aux*, *C*>
R: (Reactive) Rules

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

- *consequent* is a disjunction $consequent_1 \vee \ldots \vee consequent_n$
- *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:

$seeWolf(shep, T) \rightarrow cryWolf(shep, T+1)$

$cryWolf(shep, T) \wedge \neg help(shep, T+1) \rightarrow$
$cryWolf(shep, T+2)$

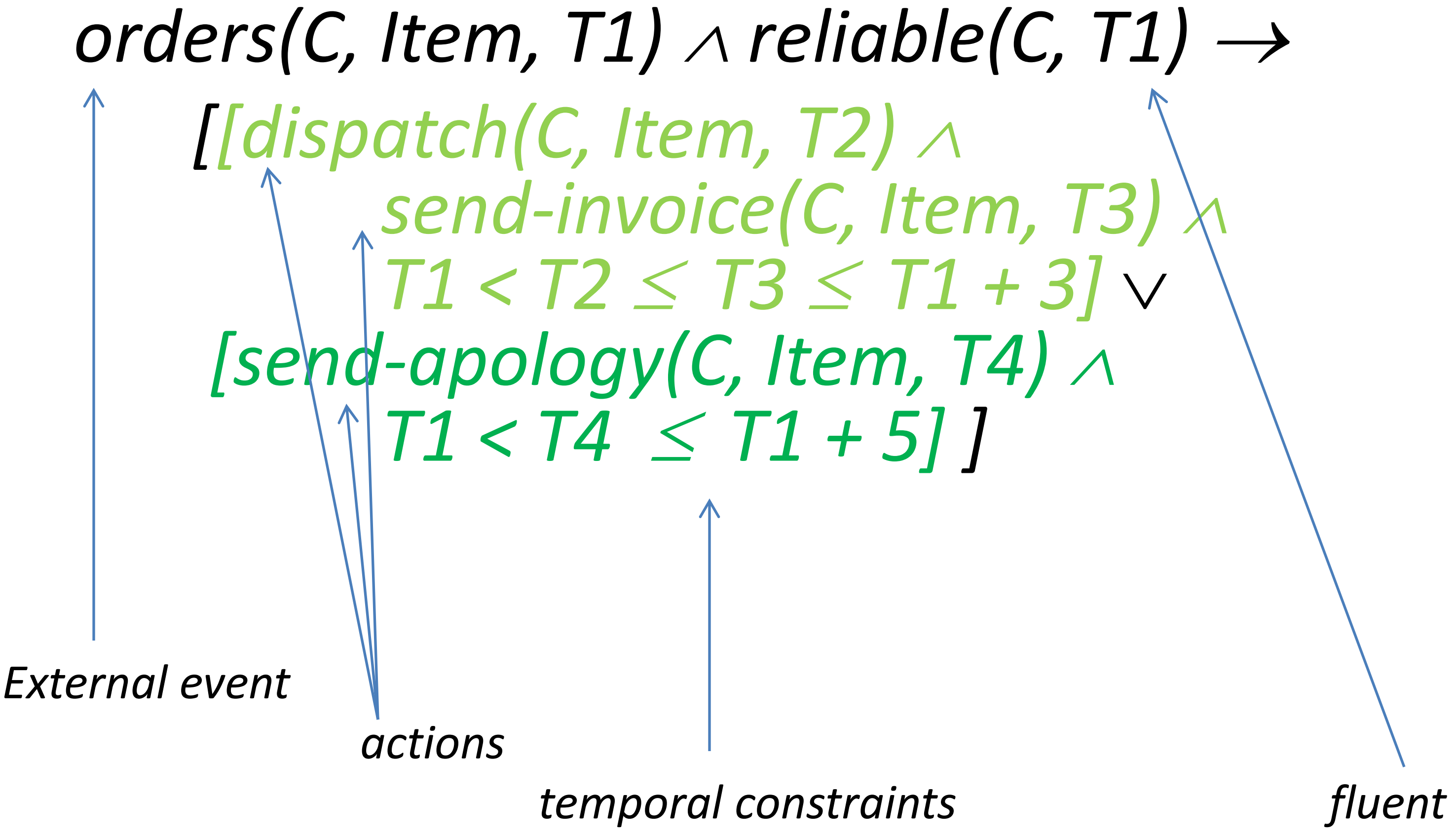
Villagers:

$cryWolf(X, T) \wedge \neg\, joker(X, T) \rightarrow help(X, T+1)$

$cryWolf(X, T1) \wedge \neg\, wolf(T1) \wedge cryWolf(X, T2) \wedge$
$\neg\, wolf(T2) \wedge T1<T2 \rightarrow assume(joker(X), T2+1)$

$initiates(assume(joker(X), joker(X)))$

Example of a more complicated Reactive Rule



$$\begin{aligned}
&\mathit{orders}(C, \mathit{Item}, T1) \wedge \mathit{reliable}(C, T1) \rightarrow \\
&\quad [[\mathit{dispatch}(C, \mathit{Item}, T2) \wedge \\
&\qquad \mathit{send\text{-}invoice}(C, \mathit{Item}, T3) \wedge \\
&\qquad T1 < T2 \leq T3 \leq T1 + 3] \vee \\
&\quad [\mathit{send\text{-}apology}(C, \mathit{Item}, T4) \wedge \\
&\qquad T1 < T4 \leq T1 + 5]\,]
\end{aligned}$$

External event

actions

temporal constraints

fluent

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} : (Integrity constraints)

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

- To constrain executability of concurrent actions

$dispatch(Cust1, Item, T) \wedge dispatch(Cust2, Item, T) \wedge Cust1 \neq Cust2 \rightarrow false$

- To require co-existence of some actions:

$leave\_house(T) \wedge \neg\ take\_keys\ (T) \rightarrow false$

- To specify preconditions of actions

$$give\_bonus(M, T) \wedge manager(M, T) \wedge \exists D\ (manages(M, D, T) \wedge loss\_making(D, T)) \rightarrow false$$

C_{post} :

initiates and *terminates* defined by (ground) atoms.

initiates(events, fluent) and

terminates(events, fluent).

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

initiates([send_invoice(C, Item)], payment_due(C, Item))

terminates([pays_invoice(C, Item)], payment_due(C, Item))

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

Aux: Auxiliary predicates defined by ground atoms.

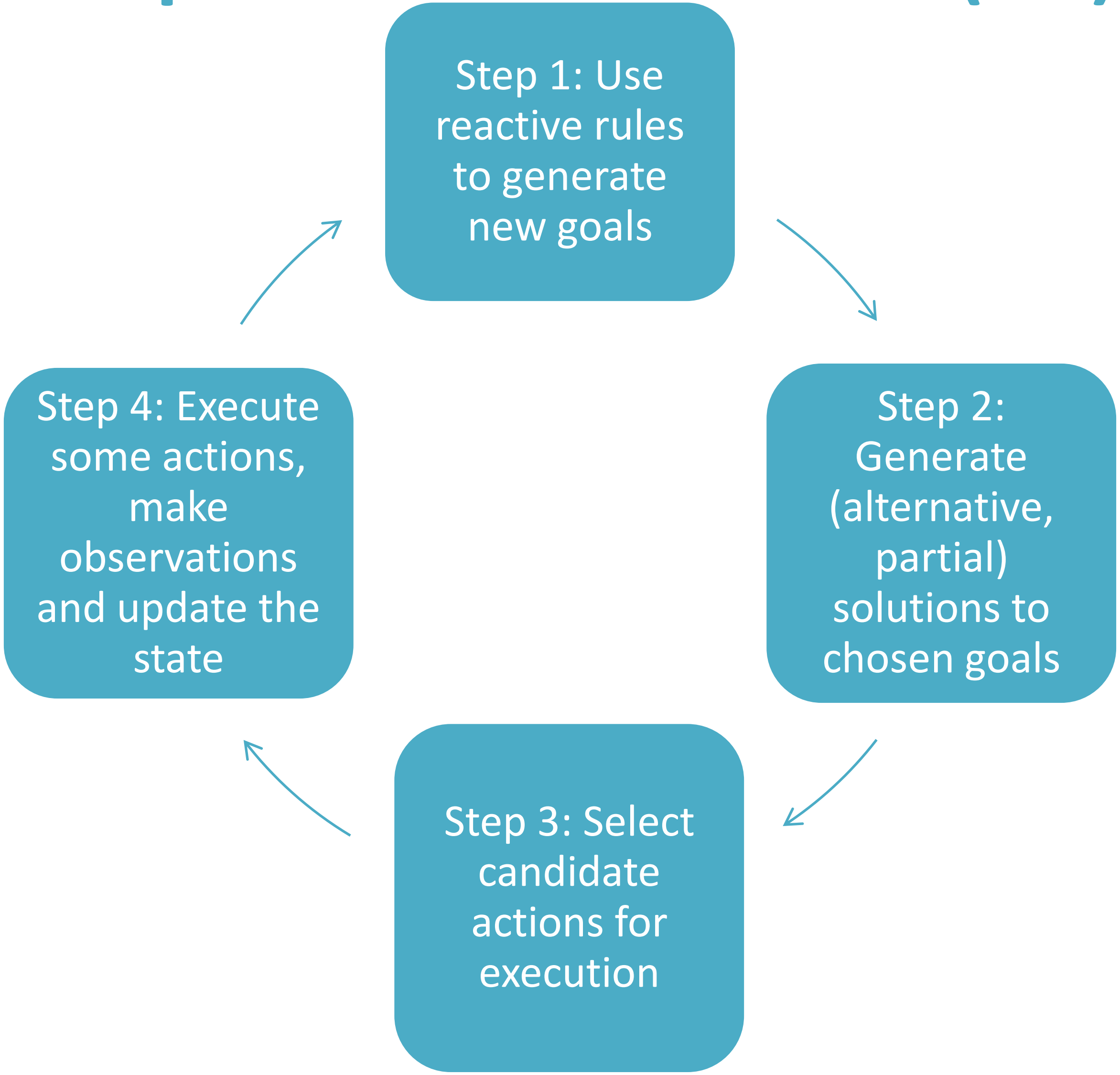
- *Time-independent predicates*, e.g.

  isa(book, product).

- *Temporal constraint predicates*, e.g.

  $i < j$ or $i \leq j$ between time points.

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

Notes on the OS: Event Stream Processing

Step 1: Use reactive rules to generate new goals

Also to recognise complex events –

Stream Processing of Events

$[ev_1(T_1) \wedge c_1(T_1) \wedge$

$ev_2(T_2) \wedge c_2(T_2) \wedge \dots \wedge$

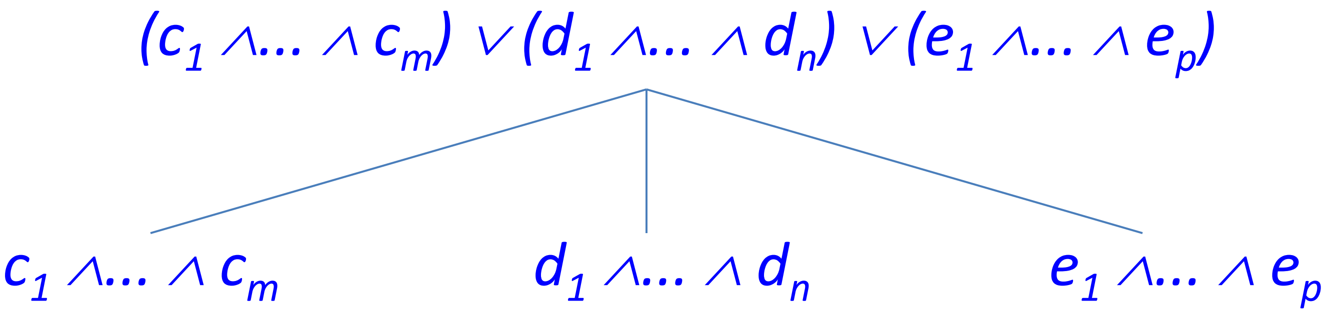
$ev_n(T_n) \wedge c_n(T_n) \wedge$

$\text{constraints on } T_1, T_2, ..T_n] \rightarrow \text{consequent}$

Notes on the OS: Goal State

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.



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

$c_1 \wedge \ldots \wedge c_m$

$d_1 \wedge \ldots \wedge d_n$

$e_1 \wedge \ldots \wedge e_p$

Notes on the OS: Implementation

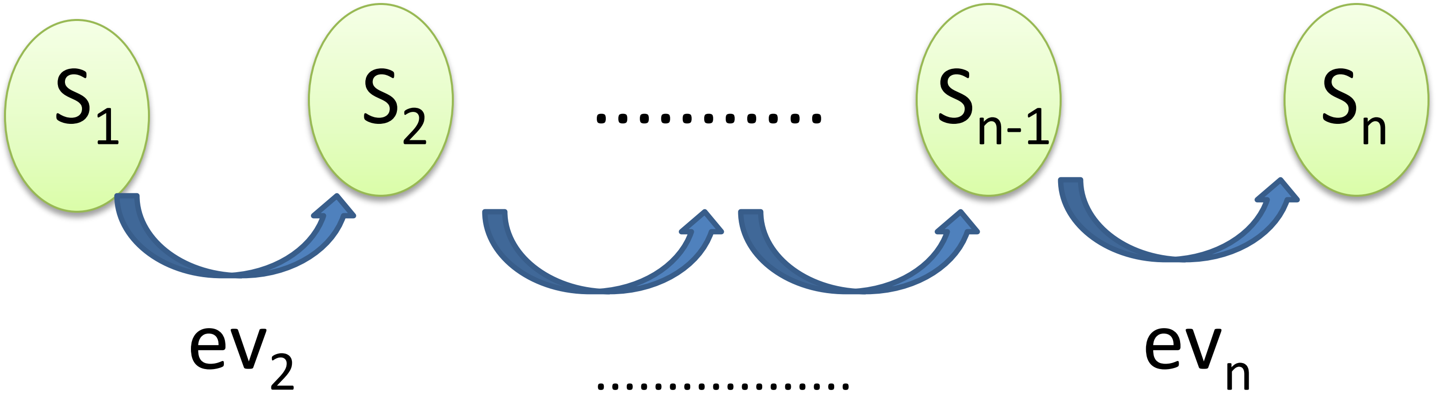
We have implemented different strategies for searching the space, e.g., based on:

- ✓ Priorities of reactive rules
- ✓ Deadlines given by the temporal constraints
- ✓ Length of time a goal has been waiting
- ✓ .....

Notes on the OS: State

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 $\quad$ $Goal\ State_0$ $\quad$ $R_0=R$

$\{ext_1 \cup acts_1\}$

S_1 $\quad$ $Goal\ State_1$ $\quad$ R_1

…

… $\quad$ …

… $\quad$ …

S_i $\quad$ $Goal\ State_i$ $\quad$ R_i

$\{ext_{i+1} \cup acts_{i+1}\}$

S_{i+1} $\quad$ $Goal\ State_{i+1}$ $\quad$ R_{i+1}

… $\quad$ … $\quad$ …

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 $<R, Aux, C>$, 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

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

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:

seeWolf(T) ∧ outdoors(T) → 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 $<R, Aux, C>$, 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).

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 $ev_0, \ldots, ev_i, \ldots$, where $ev_0 = \{\}$, let

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

$\boldsymbol{S^*} = S_0^* \cup \ldots \cup S_i^* \cup \ldots$ where

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

$\boldsymbol{ev^*} = ev_0^* \cup \ldots \cup ev_i^* \cup \ldots$

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

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

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

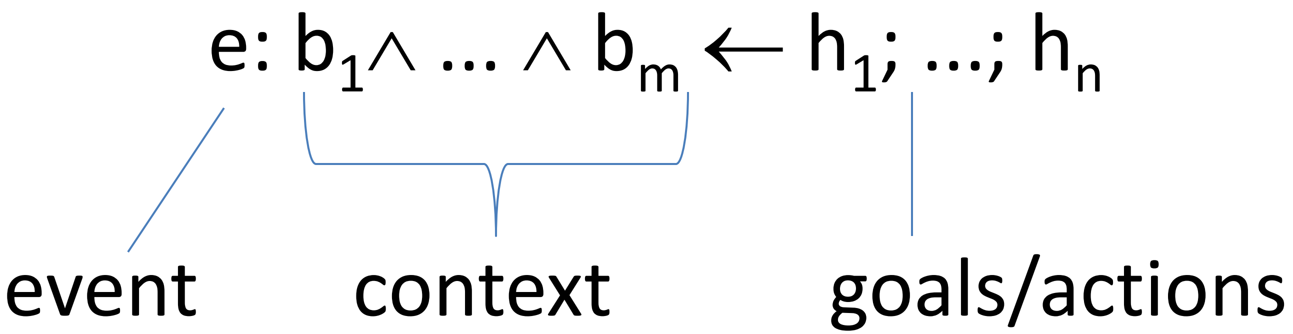
is true in $\boldsymbol{I}$.

Contents

6. Examples of KELPS/LPS Formalisations

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

BDI AgentSpeak Plans



$$e: \underbrace{b_1 \wedge \ldots \wedge b_m}_{\text{context}} \leftarrow h_1; \ldots; h_n$$

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

$$\mathit{location}(\mathit{waste}, X, T1) \wedge \mathit{location}(\mathit{robot}, X, T1) \wedge$$

$$\mathit{location}(\mathit{bin}, Y, T1)$$

$$\rightarrow \mathit{pick}(\mathit{waste}, T1+1) \wedge$$

$$\mathit{goto}(\mathit{robot}, Y, T2) \wedge T2>T1$$

$$\mathit{drop}(\mathit{waste}, T2+1)$$

$goto(robot, Y, T) \leftarrow location(robot, Y, T)$

$goto(robot, Y, T2) \leftarrow$

$location(robot, X, T1) \wedge \neg X = Y \wedge$
$adjacent(X, Z) \wedge$

$\neg location(car, Z, T1) \wedge$

$move(robot, Z, T1) \wedge$

$goto(robot, Y, T2) \wedge T1 < T2$

ECA Rules
Hospital Example

duty_nurse(N, Ward, T) $\wedge$
spot_stranger(N, Ward, T) $\rightarrow$
stream_videoCam(N, Ward, T_1) $\wedge$
set_off_alarm(N, Ward, T_2) $\wedge$
$T_1 < T+3 \wedge T_2 < T+3$

duty_nurse(N, Ward, T) $\wedge$
emergency_alert(Patient, Ward, T) $\rightarrow$
duty_head_nurse(HN, T) $\wedge$
inform(N, HN, Patient, Ward, T+1) $\wedge$
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

$aliveNeighb(C, N, T) \wedge (N<2 \vee N > 3) \wedge alive(C, T) \rightarrow$
$retract(alive(C), T+1)$

$aliveNeighb(C, N, T) \wedge N = 3 \wedge \neg\, alive(C, T) \rightarrow$
$assert(alive(C), T+1)$

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

$notify(P, S, Ser, T1) \wedge$

$\neg\ covered\_under(Ser, quality\_of\_service, T1) \rightarrow$

$\quad [[replace(S, P, Ser, T2) \wedge T2 \leq T1+3] \vee$

$\quad [refund(S, P, Ser, T3) \wedge$

$\quad pay\_penalty(S, P, Ser, \$1000, T3) \wedge T3 > T1+3]]$

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

1. R. Kowalski, F. Sadri,Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents, In Web Reasoning and Rule Systems (eds. A. Polleres and T. Swift) Springer, LNCS 5837. 2009.
2. R. Kowalski, F. Sadri An Agent Language with Destructive Assignment and Model-theoretic Semantics, In CLIMA XI - Computational Logic in Multi- Agent Systems (eds. J. Dix, G. Governatori, W. Jamroga and J. Leite) Springer, 2010.
3. R. Kowalski, F. Sadri Abductive Logic Programming Agents with Destructive Databases, In Annals of Mathematics and Artificial Intelligence, 2011.
4. R. Kowalski and F. Sadri, Teleo-Reactive Abductive Logic Programs In Festschrift for Marek Sergot.(eds: Alexander Artikis, Robert Craven, Nihan Kesim, Babak Sadighi, and Kostas Stathis), Springer, 2012.
5. R. Kowalski and F. Sadri, A Logic-Based Framework for Reactive Systems In Procedings of RuleML 2012.
6. R. Kowalski and F. Sadri, A Logical Characterization of a Reactive System Language, RuleML 2014, A. Bikakis et al. (Eds.): RuleML 2014, LNCS 8620, pp. 22-36, Springer International Publishing Switzerland , 2014.
7. R. Kowalski and F. Sadri, Reactive Computing as Model Generation, to appear in New Generation Computing, Vol. 33-1, January 2015.

Thank you for listening.

Questions