A Statelog Framework for Active Rules

- So far: abstract properties of "pure" Statelog = production rules & deductive rules
- Now: modeling of specific aspects of active rules:
  external events, delta relations, transaction control etc.

Contents

- Statelog Schema for Active Rules
- User-Defined vs. System-Defined Rules
- Integrity Constraints
- Composite Events
- Termination
- Summary

The database schema $\mathbf{R}$ is partitioned into several disjoint sets:

$$\mathbf{R} = \mathbf{edb}(\mathbf{R}) \mathbin{\dot\cup} \mathbf{idb}(\mathbf{R}) \mathbin{\dot\cup} \triangleright(\mathbf{R}) \mathbin{\dot\cup} \triangleleft(\mathbf{R}) \mathbin{\dot\cup} \boldsymbol{\delta}(\mathbf{R}) \mathbin{\dot\cup} \mathbf{ctl}(\mathbf{R})$$

- **Extensional relations $\mathbf{edb}(\mathbf{R})$:**
  - extensionally stored database relations $\widehat{=}$ usual *relational schema*
  - may be updated directly through delta relations
- **Intensional relations $\mathbf{idb}(\mathbf{R})$:**
  - derived (view) relations
  - usually not materialized
  - are updated indirectly via edb updates

- **External events $\triangleright$(R):**
  - report external event occurrences to the system
  - allowed only in rule bodies
- **External actions $\triangleleft$(R):**
  - requests to execute external actions
  - allowed only in rule heads
- **Delta relations δ(R):**
  - for every relation $R \in$ **edb**(**R**) there are delta relations **del**:R, **ins**:R, **mod**:$R \in \delta$(**R**)
  - in the head: derives update requests
  - in the body: checks for update requests
- **Control relations ctl(R):**
  - transaction control: **BOT**, **EOT**, **abort**
  - storage of net effect: **insd**:R, **deld**:R, **modd**:R (<u>ins</u>erte<u>d</u>)
  - (mostly) maintained by the system

	$\mathcal{M}[0]$				
$\triangleright e_i \rightsquigarrow$	$\mathcal{M}[0]$ **BOT**	$\cdots$	$\mathcal{M}[n]$ **EOT**		
$\triangleright e_{i+1} \rightsquigarrow$			$\mathcal{M}[n]$ **BOT**	$\cdots$	$\mathcal{M}[m]$ **EOT** $= \mathcal{M}[\$]$
$\cdots$					$\cdots$

Classification of States

state	**BOT**	**EOT**
initial	1	0
intermediate	0	0
current final	0	1
past final	1	1

- $\Rightarrow$ **logically** the Statelog model is *add-only*
- *fixed window property & Statelog normal form*
  $\Rightarrow$ at most two states have to be maintained at a time

- **User-defined rules** provide the user with predefined high-level update requests & logical views
- **System-defined rules** relieve the user from transactional control, explicit handling of states etc.

Types of User-Defined Rules

Views:	$[S]\ V(\bar{X}) \leftarrow \ldots$	$V \in$ **idb**(**R**)
Change Requests:	$[S]$ **ins**:$R(\bar{X}) \leftarrow \ldots$	
	$[S]$ **del**:$R(\bar{X}) \leftarrow \ldots$	$R \in$ **edb**(**R**)
	$[S]$ **mod**:$R(\bar{X}/\bar{X}') \leftarrow \ldots$	
External Actions:	$[S]\ \triangleleft A(\bar{X}) \leftarrow \ldots$	$\triangleleft A \in \triangleleft$(**R**)
Transaction Control:	$[S]$ **abort** $\leftarrow \ldots$	**abort** $\in$ **ctl**(**R**)

$\Rightarrow$ all user-defined rules are **local**

$\Rightarrow$ **state terms may be omitted from user programs**

Example:

[\$] on(red)	$\leftarrow$	*% actual database state*
ins:on(yellow)	$\leftarrow$	$\rhd$tick, on(red), $\neg$ on(yellow)
del:on(red), **del**:on(yellow), **ins**:on(green)	$\leftarrow$	$\rhd$tick, on(red), on(yellow)
del:on(green), **ins**:on (yellow)	$\leftarrow$	$\rhd$tick, on(green)
del:on(yellow), **ins**:on(red)	$\leftarrow$	$\rhd$tick, on(yellow), $\neg$ on(red)
$\lhd$on(X)	$\leftarrow$	on(X)
$\lhd$off(X)	$\leftarrow$	$\neg$ on(X)

Possible trace:

[\$] on(red) *% current (= previous final) state*
...
[\$] $\rhd$ tick $\Rightarrow$ **BOT**, **ins**:on(yellow), **running** *% current state is extended*
$\leadsto$ *% transition*
[\$+1] on(red), on(yellow), $\neg$ **running**, **EOT** *% new final state*
...

External Events:

$$[S]\ \textbf{BOT} \leftarrow\ \triangleright\mathrm{E}(\bar{X})$$

Frame Rules:

$$\begin{array}{lll} [S{+}1]\ \mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{ins}{:}\mathrm{R}(\bar{X}),\ \neg\,\textbf{EOT} \\ [S{+}1]\ \mathrm{R}(\bar{X}) & \leftarrow & [S]\ \mathrm{R}(\bar{X}),\ \neg\,\textbf{del}{:}\mathrm{R}(\bar{X}),\ \neg\,\textbf{EOT} \end{array}$$

Protocol Relations:

$$\begin{array}{llll} [S{+}1]\ \textbf{insd}{:}\mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{ins}{:}\mathrm{R}(\bar{X}),\ \neg\,\mathrm{R}(\bar{X})\ . & \textit{\% store net effect...} \\ [S{+}1]\ \textbf{insd}{:}\mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{insd}{:}\mathrm{R}(\bar{X}),\ \neg\,\textbf{del}{:}\mathrm{R}(\bar{X}) & \textit{\% ... of insertions} \\ [S{+}1]\ \textbf{deld}{:}\mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{del}{:}\mathrm{R}(\bar{X}),\ \mathrm{R}(\bar{X})\ . & \textit{\% store net effect...} \\ [S{+}1]\ \textbf{deld}{:}\mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{deld}{:}\mathrm{R}(\bar{X}),\ \neg\,\textbf{ins}{:}\mathrm{R}(\bar{X}) & \textit{\% ... of deletions} \\ [S{+}1]\ \mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{abort},\ \textbf{deld}{:}\mathrm{R}(\bar{X}). & \textit{\% undo deletions} \\ [S{+}1]\ \mathrm{R}(\bar{X}) & \leftarrow & [S]\ \textbf{abort},\ \mathrm{R}(\bar{X}),\ \neg\,\textbf{insd}{:}\mathrm{R}(\bar{X}). & \textit{\% undo insertions} \\ [S{+}1]\ \textbf{EOT} & \leftarrow & [S]\ \textbf{abort}. & \textit{\% signal end of transaction} \end{array}$$

Transaction Control:

$[S]$ **running** $\leftarrow$ $[S]$ **ins**:R($\bar{X}$), $\neg$ R($\bar{X}$).
$[S]$ **running** $\leftarrow$ $[S]$ **del**:R($\bar{X}$), R($\bar{X}$).

$[S]$ **EOT** $\leftarrow$ $[S]$ **BOT**, $\neg$ **running**.
$[S{+}1]$ **EOT** $\leftarrow$ $[S]$ **running**, $[S{+}1]$ $\neg$ **running**.

Conflict Resolution:

What if conflicting update requests **ins**:p $\wedge$ **del**:p occur?

$[S]$			$[S{+}1]\ p$				
p	**ins**:p	**del**:p	INS	DEL	NOP	DOP	ABORT
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*
false	*false*	*true*	*false*	*false*	*false*	*false*	*false*
false	*true*	*false*	*true*	*true*	*true*	*true*	*true*
false	**true**	**true**	**true**	**false**	**false**	**true**	$\perp$
true	*false*	*false*	*true*	*true*	*true*	*true*	*true*
true	*false*	*true*	*false*	*false*	*false*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*	*true*
true	**true**	**true**	**true**	**false**	**true**	**false**	$\perp$

% Priority to Deletions (DEL):

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

% Priority to Insertions (INS):

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

% No-Operation Policy (NOP):

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

% Do-Operation Policy (DOP):

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, \neg\, R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

% Abortion Policy (ABORT):

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}).$

$[S{+}1]\; R(\bar{X}) \;\leftarrow\; [S]\; R(\bar{X}),\, \neg\, \textbf{del}{:}R(\bar{X}).$

$[S]\; \textbf{abort} \;\leftarrow\; [S]\; \textbf{ins}{:}R(\bar{X}),\, \textbf{del}{:}R(\bar{X}).$

Static Constraints

All usual constraints like e.g. *functional, join, inclusion* dependencies can be expressed as FO-queries (cf. [AHV95]) $\Rightarrow$ also as local Statelog rules (recall that FO $<$ S-Datalog$^{\neg}$ wrt. expressive power)

Example

- inclusion dependency id: emp[DNo] $\subseteq$ dept[DNo]
- in FO: φ_{id} : $\forall E, Sal, D\ (\text{emp}(E, Sal, D) \rightarrow \exists M\ \text{dept}(D, M))$
- thus $\neg\,\varphi_{id} \Leftrightarrow \exists E, Sal, D\ (\text{emp}(E, Sal, D) \wedge \neg\,\exists M\ \text{dept}(D, M))$
- in Statelog:

false	$\leftarrow$	emp(E,Sal,D), ¬ ex_mgr_of_dept(D)
ex_mgr_of_dept(D)	$\leftarrow$	dept(D,M)

Dynamic Constraints

Temporal constraints on the **database history** ($\mathcal{M}[0]\,,\ldots,\mathcal{M}[\$]$) can be expressed as **PFOTL** (*past first-order temporal logic*) formulas [Cho95]:

Example: *Salaries should never decrease!*

in PFOTL:

$$\neg\,\exists E, Sal_1, Sal_2\ (\blacklozenge emp(E, Sal_1) \wedge emp(E, Sal_2) \wedge Sal_1 > Sal_2)$$

in Statelog:

$[S]$ **false** $\leftarrow$ **sp**_emp(E,Sal1), emp(E,Sal2), Sal1 $>$ Sal2

$[S{+}1]$ **sp**_emp(E,Sal) $\leftarrow$ $[S]$ emp(E,Sal)
$[S{+}1]$ **sp**_emp(E,Sal) $\leftarrow$ $[S]$ **sp**_emp(E,Sal)

$\bullet F \;\widehat{=}\;$ *previously* $F \;\widehat{=}$

$$\boxed{[S{+}1]\ \mathsf{prev}_F \leftarrow [S]\ F}$$

$\blacklozenge F \;\widehat{=}\;$ *sometime in the past* $F \;\widehat{=}$

$$\boxed{\begin{array}{lcl} [S{+}1]\ \mathbf{sp}_F & \leftarrow & [S]\ F \\ [S{+}1]\ \mathbf{sp}_F & \leftarrow & [S]\ \mathbf{sp}_F \end{array}}$$

G **since** $F \;\widehat{=}\;$ *sometime in the past* F*, and* G *since then* $\widehat{=}$

$$\boxed{\begin{array}{lcl} [S{+}1]\ \mathbf{since}_{G,F} & \leftarrow & [S]\ F,\ [S{+}1]\ G \\ [S{+}1]\ \mathbf{since}_{G,F} & \leftarrow & [S]\ \mathbf{since}_{G,F}\ ,\ [S{+}1]\ G \end{array}}$$

$\blacksquare F \;\widehat{=}\;$ *always in the past* $F \;\widehat{=}\; \neg \blacklozenge \neg F$

NB: operators may be **relativized** to **final states**!

Sequence: $E(X, Y) = (F(X)\ ;\ G(Y))$ *"E when G after F"*

unrestricted context:

$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ F(X)
$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ **detd**_F(X)
$[S]$ E(X,Y) $\leftarrow$ $[S]$ **detd**_F(X), G(Y)

$\Rightarrow$ equivalent to $G(Y) \wedge \blacklozenge F(X)$

cumulative context:

$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ F(X)
$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ **detd**_F(X), $\neg$ E(_,_)
$[S]$ E(X,Y) $\leftarrow$ $[S]$ **detd**_F(X), G(Y)

recent context:

$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ F(X)
$[S{+}1]$ **detd**_F(X) $\leftarrow$ $[S]$ **detd**_F(X), $\neg$ F(_), $\neg$ E(_,_)
$[S]$ E(X,Y) $\leftarrow$ $[S]$ **detd**_F(X), G(Y)

$\Rightarrow$ some rules are not stratified but **state-stratified**!

chronicle context:

$$
\begin{array}{llll}
[S]\ \textbf{detd\_}\mathsf{F}(\mathsf{X}_T) & \leftarrow & [S]\ \mathsf{F}(\mathsf{X}),\ T = S & \textit{\% not pure Statelog!} \\
[S{+}1]\ \textbf{detd\_}\mathsf{F}(\mathsf{X}_T) & \leftarrow & [S]\ \textbf{detd\_}\mathsf{F}(\mathsf{X}_T),\ [S{+}1]\ \neg\,\mathsf{E}(\mathsf{X}_T,\_) & \\
[S{+}1]\ \mathsf{E}(\mathsf{X},\mathsf{Y}) & \leftarrow & [S{+}1]\ \mathsf{G}(\mathsf{Y}),\ [S]\ \textbf{detd\_}\mathsf{F}(\mathsf{X}_T),\ \neg\,\textbf{nfirst\_}\mathsf{F}(\mathsf{X}_T) & \\
[S]\ \textbf{nfirst\_}\mathsf{F}(\mathsf{X}_{T1}) & \leftarrow & [S]\ \textbf{detd\_}\mathsf{F}(\mathsf{Y}_{T2}),\ \textbf{detd\_}\mathsf{F}(\mathsf{X}_{T1}),\ T2 < T1 &
\end{array}
$$

$\Rightarrow$ **FIFO** processing; requires **timestamping**

$\Rightarrow$ an unbounded number of timestamps is needed

recent context:

with **LIFO** strategy (i.e. remember unused events) $\Rightarrow$ unbounded number of timestamps

(other event constructors like *disjunction*, *conjunction*, *any*, ... may be easily specified in Statelog)

Theorem (Termination).

- *at **compile-time**, termination is undecidable, i.e. whether a program always/sometimes terminates (use Trakhtenbrot's Theorem)*
- *at **run-time**, i.e. given P and $\mathcal{D}$, deciding termination is $\boldsymbol{PSPACE}$-complete*

How to detect termination at run-time?

Clearly, if $\mathcal{M}_P[\$] = \mathcal{M}_P[\$+1] \Rightarrow$ final state reached[1]

$\Rightarrow$ for all $R \in$ **edb**(**P**):

$$\boxed{\begin{array}{llll} [S] & \textbf{running} & \leftarrow & [S]\ \textbf{ins}{:}\mathsf{R}(\bar{X}),\ \neg\,\mathsf{R}(\bar{X}). \\ [S] & \textbf{running} & \leftarrow & [S]\ \textbf{del}{:}\mathsf{R}(\bar{X}),\ \mathsf{R}(\bar{X}). \end{array}}$$

[1] W.l.o.g. $P \in$ NF-Statelog, i.e. in normal form

How to detect nontermination at run-time?

Problem: $\mathcal{M}_P$ does not terminate iff for all $n \geq \$$: $\mathcal{M}_P[n] \neq \mathcal{M}_P[n{+}1]$

- **outside** of Statelog: there are at most $n_0 = |2^{\mathcal{B}_P}|$ different database states
  $\Rightarrow \mathcal{M}_P$ terminates iff $\mathcal{M}_P[n_0] = \mathcal{M}_P[n_0{+}1]$
- **within** Statelog: use delaying technique (similar to [AS91])

Theorem. *For every P, there is a (transaction) equivalent $P^{\downarrow}$ s.t.*

$$\begin{array}{lcl} \mathcal{M}_{P^{\downarrow} \cup \mathcal{D}}[\$] \models \textbf{\textit{terminates}} & \Leftrightarrow & \mathcal{M}_{P \cup \mathcal{D}} \textit{ terminates for } \mathcal{D}. \\ \mathcal{M}_{P^{\downarrow} \cup \mathcal{D}}[\$] \models \textbf{\textit{diverges}} & \Leftrightarrow & \mathcal{M}_{P \cup \mathcal{D}} \textit{ does not terminate for } \mathcal{D}. \end{array}$$

- NB: $P^{\downarrow}$ solves the problem only for a **given** $\mathcal{D}$, i.e. at **run-time**.
  (whether $P^{\downarrow}$ always/sometimes yields "**terminates**" is undecidable)

Recursion in Statelog

- **local recursion:**
  rule execution **within a state** corresponds to evaluating a stratified Datalog$^{\neg}$ program
  $\Rightarrow$ termination is guaranteed
- **progressive recursion:**
  recursive triggering of rules **across** different states may lead to oscillating update requests $\Rightarrow$ nontermination:

$$\boxed{\begin{array}{lll} \textbf{ins}{:}\mathsf{p} & \leftarrow & \neg\,\mathsf{p} \\ \textbf{del}{:}\mathsf{p} & \leftarrow & \mathsf{p} \end{array}}$$

Guarded Rules

- a rule r is **guarded** if an external event $\triangleright E$ occurs positively in the body

$\Rightarrow$ r is executed **at most once** within a transaction

$\Rightarrow$ if all user-defined update rules are guarded (cf. traffic-light example), termination is guaranteed

Problem: rather restrictive, **no progressive recursion**

$\mathcal{M}_P$ *inflationary* $\Rightarrow$ termination guaranteed; however **no deletions**!

Δ-Monotonic Rules

Basic idea: avoid oscillating update requests, i.e. repeated insertion and deletion of the same facts

Different granularities:

let $R \in$ **edb**(**R**):

1. the rules may not contain both **ins**:R and **del**:R in rule heads (*compile-time check*)
2. both **ins**:R and **del**:R are allowed;
   at run-time it is checked whether initially, R increases or decreases
   $\Rightarrow$ abort if complementary action is taken (*run-time check, relation level*)
3. like (2.) but at the *tuple level*:

$$\boxed{\begin{array}{llll} [S] & \textbf{abort} & \leftarrow & [S]\ \textbf{insd}{:}R(\bar{X}),\ \textbf{del}{:}R(\bar{X}) \\ [S] & \textbf{abort} & \leftarrow & [S]\ \textbf{deld}{:}R(\bar{X}),\ \textbf{ins}{:}R(\bar{X}) \end{array}}$$

Theorem. P *Δ-monotonic* $\Rightarrow$ P *always terminates within* $\boldsymbol{PTIME}$

Deductive Rules "higher" level		*Production Rules*			*Active Rules* "lower" level
stratified, well-founded Datalog$^{\neg}$	RDL procedural Datalog$^{\neg}$	A-RDL	Ariel	Starburst	Postgres

- Distinctive features of rules: active, production, deductive
- Integration in Statelog:
  - logical, set-oriented semantics for queries & updates $\Rightarrow$ (re)active behavior
  - static & *dynamic* IC checking and *enforcement*
  - composite event detection
  - handling of termination
  - expressive power, complexity
- more low-level or procedural features $\Rightarrow$ add more "control", e.g. **nested transactions** [LML96]
- some applications
  - (re)active db-programming
  - view maintenance (e.g. DRED [GMS93])
  - specification/implementation of *referential actions* [LMR96, LML97]

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[AS91] S. Abiteboul and E. Simon. Fundamental Properties of Deterministic and Nondeterministic Extensions of Datalog. *Theoretical Computer Science*, 78(1):137–158, 1991.

[Cho95] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, 1995.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *Proc. ACM SIGMOD Conf.*, page 157, Washington, DC, May 1993.

[LML96] B. Ludäscher, W. May, and G. Lausen. Nested Transactions in a Logical Language for Active Rules. In D. Pedreschi and C. Zaniolo, editors, *Proc. Intl. Workshop on Logic in Databases (LID)*, number 1154 in LNCS, pages 196–222, San Miniato, Italy, 1996. Springer.

[LML97] B. Ludäscher, W. May, and G. Lausen. Referential Actions as Logical Rules. In *Proc. 16th ACM Symposium on Principles of Database Systems*, Tucson, Arizona, 1997.

[LMR96] B. Ludäscher, W. May, and J. Reinert. Towards a Logical Semantics for Referential Actions in SQL. In *Proc. 6th Intl. Workshop on Foundations of Models and Languages for Data and Objects: Integrity in Databases*, Dagstuhl, Germany, 1996.