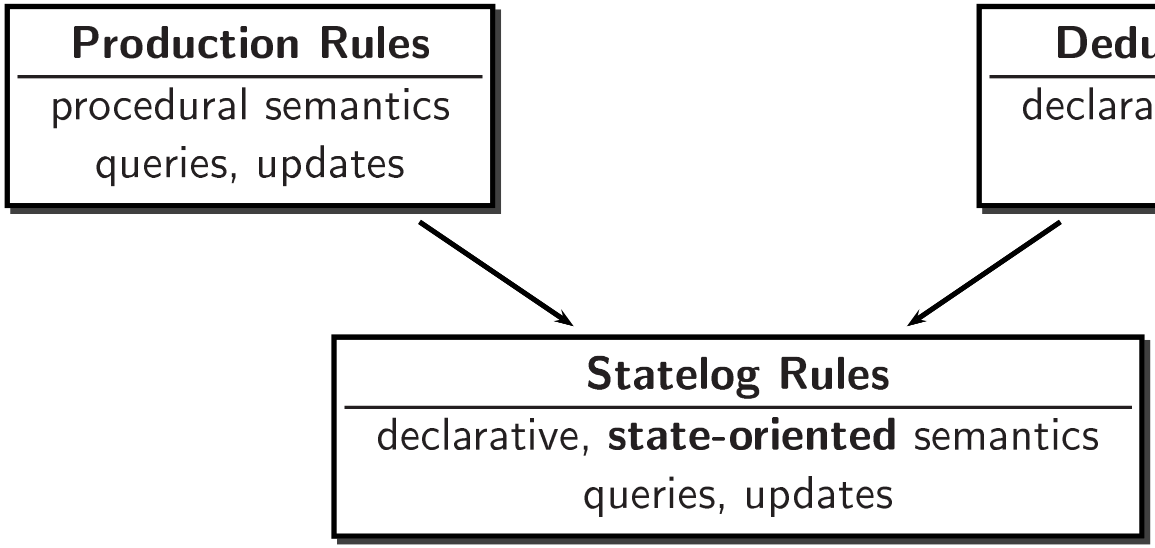
(Pure) Statelog

Contents

- Motivation: Unifying Active and Deductive Rules
- Statelog: Datalog$^{\neg}$ + States
- Syntax, Semantics
- Canonical State-Stratified Model $\mathcal{M}_P$
- Theoretical Aspects: Complexity, Termination, Expressiveness
- Normal Forms

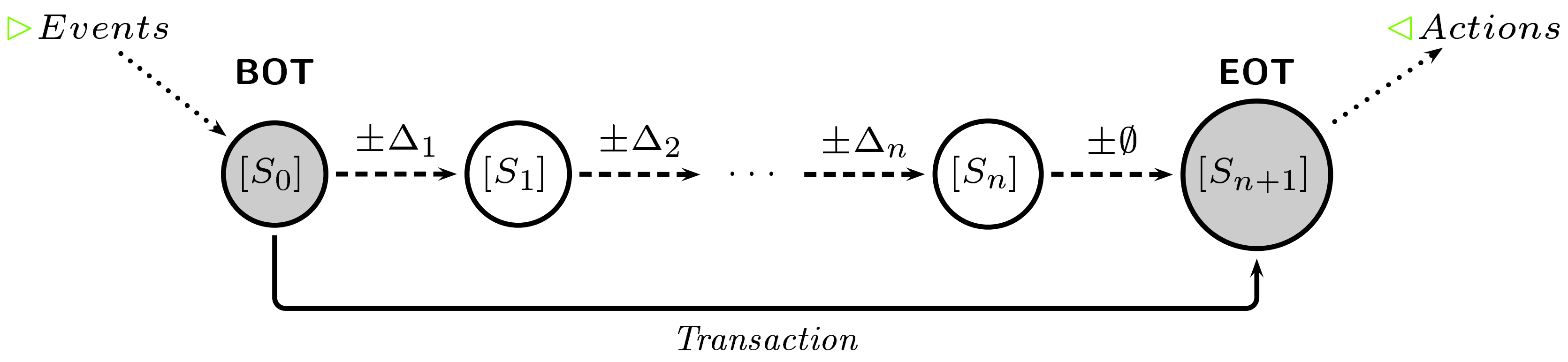


Recall the different "readings" of rules:

- **Deductive Rules:** if *premise* then *conclusion*
  $\Rightarrow$ reasoning process: knowledge about the **same state** is only increased; stateless language
- **Production rules:** if *condition* then *action*
  $\Rightarrow$ defines a **new database state** S' from the current state S; states are **implicit** in the production rule semantics

$\Rightarrow$ incorporate states into the logical language $\Rightarrow$ "local" recursion (e.g. transitive closure) can be distinguished from "progressive" recursion (e.g. cascaded delete requests)

Database Evolution: A Transaction and Intermediate Transitions:



- Local **query rules** are used to query the current state $[S_0]$ without changing it.
- Triggered by the occurrence of **external events** $\triangleright e(\bar{x})$, a **transaction** begins with the **transition** to the intermediate state $[S_1]$.
- More rules may be triggered resulting in further states $[S_2]$, $[S_3]$, ... until a new final state $[S_{n+1}]$ is reached.

Queries: $\boxed{[S]\ \text{tc(X,Y)} \leftarrow \text{tc(X,Z), tc(Z,Y)}}$

Integrity Constraints: $\boxed{[S]\ \textbf{abort} \leftarrow \ldots}$

Event detection: $\boxed{[S]\ \textbf{del}\text{:dept(D,M)} \leftarrow \triangleright\textbf{del}\text{:dept(D,M)}}$

Update propagation:

% emp.D REFERENCES dept.D ON DELETE CASCADE

$[S]$ **del**:emp(E,Sal,D) $\leftarrow$ **del**:dept(D,_), emp(E,Sal,D)

Frame Rules:

$[S{+}1]$ dept(D,M) $\leftarrow$ $[S]$ dept(D,M), $\neg$ **del**:dept(D,M)

$[S{+}1]$ dept(D,M) $\leftarrow$ $[S]$ **ins**:dept(D,M)

Integrity Constraints: $\boxed{[S]\ \textbf{abort} \leftarrow \textbf{ins}\text{:dept(D,M), }\textbf{del}\text{:dept(D,M)}}$

Here we use a **shorthand notation**, i.e. state terms $[S{+}k]$ may be "factored out" from a sequence of literals. E.g. the last rule expands to:

$\boxed{[S]\ \textbf{abort} \leftarrow [S]\ \textbf{ins}\text{:dept(D,M), } [S]\ \textbf{del}\text{:dept(D,M)}}$

Statelog extends Datalog$^\neg$ syntax by **state terms** of the form $[S{+}k]$ where S is the distinguished **state variable**, and $k \in \mathbb{N}_0$ is an **offset**.

- a **Statelog rule** r has the form:

$$\boxed{[S{+}k_0]\,H \;\leftarrow\; [S{+}k_1]\,B_1,\;\ldots\;,[S{+}k_n]\,B_n}$$

  where H is an atom, B_i are atoms or negated atoms
- r is **local** if $k_0 = k_i$, resp. **progressive** if $k_0 \geq k_i$, for $i = 1, \ldots k$
- a **Statelog program** comprises finitely many rules and possibly a finite **initial database state** $\mathcal{D}$ (as a set of facts $[0]\,p(a,b), [0]\,q(c), \ldots)$

Requirement: **all Statelog rules must be progressive!**

Rationale:

- *states* $\approx$ *snapshots at certain time instants* $\Rightarrow$ evaluation of the current state may neither change the (db-)history, nor depend on unknown future states
- all states can be computed successively by "sliding" a window of fixed width over the infinite history

Semantics: Statelog Structures

- a **Statelog structure** $\mathcal{I}$ is an infinite sequence $(\mathcal{I}[0]\,, \mathcal{I}[1]\,, \mathcal{I}[2]\,, \ldots)$
- each **database (state)** $\mathcal{I}[k]$ is finite
- $\mathcal{I}[0]$ is the **initial state** of $\mathcal{I}$
- if $\mathcal{I}$ becomes stationary, i.e. there is some n such that

$$\mathcal{I}[n] \;=\mathcal{I}[n{+}1] \;=\mathcal{I}[n{+}2] \;=\ldots$$

  then
  - $\mathcal{I}$ is called **terminating**
  - $\mathcal{I}[n]$ is the **final state** of $\mathcal{I}$
  - notation: $\mathcal{I}[\$]$ denotes the final state (if it exists)
- $\mathcal{I}$ models the **database evolution** from $\mathcal{I}[0]$ to $\mathcal{I}[\$]$
- $\mathcal{I}$ has the **fixed-window property** if there is a function f s.t.

$$\mathcal{I}[n] \;= f(\mathcal{I}[n{-}w]\,, \ldots, \mathcal{I}[n{-}1]\,)$$

  for some fixed window width w and all $n \geq n_0$
  $\Rightarrow$ eventually $\mathcal{I}$ can be computed in a state-by-state manner looking at no more than w previous states

A Statelog structure may be presented in several equivalent ways:

Snapshot view: $\mathcal{I} : \ \mathbb{N}_0 \rightarrow 2^{\mathcal{B}_P}$

$$0 \mapsto \{p(a), q(b), \ldots\}, 1 \mapsto \{p(b), q(b), \ldots\}, 2 \mapsto \{\ldots\}, \ldots$$

Timestamp view: $\mathcal{I} : \ \mathcal{B}_P \rightarrow 2^{\mathbb{N}_0}$

$$p(a) \mapsto \{0, 3, 5, \ldots\}, p(b) \mapsto \{7\}, \ldots$$

Reified (model-theoretic) view: $\mathcal{I} \subseteq \mathbb{N}_0 \times \mathcal{B}_P$

$$\{p(0, a), q(0, b), p(1, b), q(1, b), \ldots\}$$

Statelog syntax:

$$[S{+}1]\ \text{emp(E,Sal)} \ \leftarrow\ [S]\ \text{emp(E,Sal)},\ [S]\ \neg\,\textbf{del}\text{:emp(E,Sal)}$$

Templog-like syntax:

$$\Box(\circ\text{emp(E,Sal)} \ \leftarrow\ \text{emp(E,Sal)},\ \neg\,\textbf{del}\text{:emp(E,Sal)})$$

Reified syntax:

$$\text{emp}(S,\ \text{E,Sal}) \ \leftarrow\ \text{emp}(S,\ \text{E,Sal}),\ \neg\,\textbf{del}\text{:emp}(S,\ \text{E,Sal}),\ \textbf{state}(S)$$

where **state** defines the possible state terms ($\cong \mathbf{N}_0$) as follows:

$$\textbf{state}(0) \ \leftarrow$$
$$\textbf{state}(\text{succ}(S)) \ \leftarrow\ \textbf{state}(S)$$

- For a Statelog program P, the **reified version** is denoted by $P^\star$

$\Rightarrow$ using $P^\star$ the well-established declarative semantics for logic programs can be applied to Statelog

A rule

$$head(S, \bar{X}) \leftarrow body(S, \bar{Y})$$

stands for the universally closed formula

$$\forall S \; \forall \bar{X} \forall \bar{Y} \; (head(S, \bar{X}) \leftarrow body(S, \bar{Y}))$$

Satisfaction relation:

$$\begin{array}{rcl}
\mathcal{I} \models \forall S \; \varphi & :\Leftrightarrow & \mathcal{I} \models \varphi\{S/n\} \text{ for all } n \in \mathbb{N}_0 \\
\mathcal{I} \models \forall \bar{X} \; \varphi & :\Leftrightarrow & \mathcal{I} \models \varphi\{\bar{X}/\bar{d}\} \text{ for all } \bar{d} \in \mathcal{U} \\
\mathcal{I} \models \varphi \; \leftarrow \; \psi & :\Leftrightarrow & \mathcal{I} \models \psi \text{ implies } \mathcal{I} \models \varphi \\
\cdots & :\Leftrightarrow & \cdots \\
\mathcal{I} \models [n] \, \varphi & :\Leftrightarrow & \mathcal{I}[n] \models \varphi
\end{array}$$

Which is the "right" / "appropriate" semantics for Statelog programs?

$P^\star$ induces the declarative semantics of a Statelog program P:

- $P^\star$ stratified
  $\Rightarrow$ unique intended model, but too restrictive (see below)
- well-founded model of $P^\star$
  $\Rightarrow$ unique intended (but 3-valued) model
- stable models of $P^\star$
  problems:
  - complexity within a state (**NP** or above)
  - $P^\star$ may have 0, 1, many stable models

  $\Rightarrow$ not every program has a model, sets of models have to be maintained, nondeterministic semantics

$\Rightarrow$ use **state-stratification**, i.e. ensure that **within a single state** all rules of P are stratified

Example:

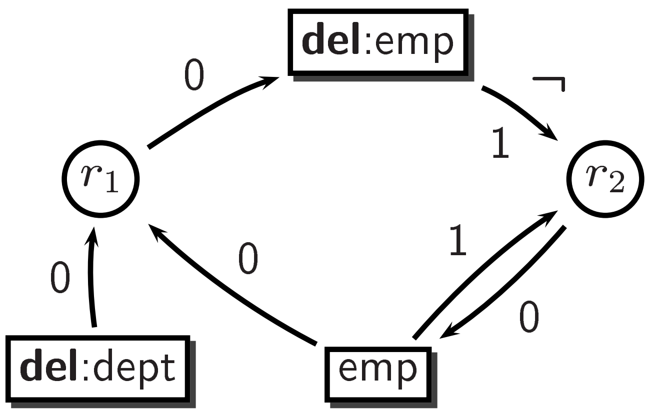
P:

$[S{+}1]$ **del**:emp(E,Sal,D)	$\leftarrow$	$[S]$ **del**:dept(D,_), emp(E,Sal,D)	%	(*r1*)
$[S{+}1]$ emp(E,Sal,D)	$\leftarrow$	$[S]$ emp(E,Sal,D), $\neg$ **del**:emp(E,Sal,D)	%	(*r2*)

In $P^\star$, r_1 depends negatively on itself $\Rightarrow$ **not stratified**. But the dependency is wrt. **different** states!

$\Rightarrow$ **state-stratified**

a **labeled dependency graph:**



- P is **state-stratified** if the labeled dependency graph of P contains no local cycle with a negative edge.

Theorem. *Let P be a progressive and rectified*[1] *Statelog program. Then*

$$P \text{ is state-stratified iff } P^\star \text{ is locally stratified}$$

$\Rightarrow$ every state-stratified program P has a **unique intended model** denoted by $\mathcal{M}_P$:

$$\mathcal{M}_P = (\mathcal{M}_P[0]\,, \mathcal{M}_P[1]\,, \mathcal{M}_P[2]\,, \ldots)$$

$\Rightarrow$ in the sequel: P is state-stratified, $\mathcal{M}_P$ the canonical model of P

- If the final state $\mathcal{M}_P[\$]$ exists, P is said to **terminate** for $\mathcal{D}$.

(We assume that $P = P' \cup \mathcal{D}$, i.e. P comprises rules P' and the initial database state as a set of facts $[0]\,p(a,b)$, $[0]\,q(c), \ldots$)

[1] the *rules* of P do not contain constants

Production Rules

- **NI-Datalog$^{\neg}$:** $\boxed{H \leftarrow B} \Longrightarrow \boxed{[S{+}1]\ H \leftarrow [S]\ B}$
- **I-Datalog$^{\neg}$:** $\boxed{H \leftarrow B} \Longrightarrow \boxed{\begin{array}{l}[S{+}1]\ H \leftarrow [S]\ B \\ [S{+}1]\ H \leftarrow [S]\ H\end{array}}$
- **Datalog$^{\neg\neg}$:**

  $\boxed{H \leftarrow B} \Longrightarrow \boxed{\begin{array}{l}[S]\ \textbf{ins}{:}H \leftarrow [S]\ B \\ [S{+}1]\ H \leftarrow [S]\ \textbf{ins}{:}H\end{array}}$

  $\boxed{\neg H \leftarrow B} \Longrightarrow \boxed{\begin{array}{l}[S]\ \textbf{del}{:}H \leftarrow [S]\ B \\ [S{+}1]\ H \leftarrow [S]\ H, [S]\ \neg\,\textbf{del}{:}H\end{array}}$

Deductive Rules

- **S-Datalog$^{\neg}$:** $\boxed{H \leftarrow B} \Longrightarrow \boxed{[S]\ H \leftarrow [S]\ B}$
- **WF-Datalog$^{\neg}$:** $\boxed{H \leftarrow B, \neg C} \Longrightarrow \boxed{[S{+}1]\ H \leftarrow [S{+}1]\ B, [S]\neg\, C}$

  (or use the well-founded model $\mathcal{W}_P$, extending $\mathcal{M}_P$),

- a Statelog structure $\mathcal{M} = (\mathcal{M}[0]\,, \mathcal{M}[1]\,, \ldots)$ is **ultimately periodic** if there exist $n_0, T \in \mathbb{N}_0$ s.t.

$$\mathcal{M}[n] = \mathcal{M}[n{+}T] \text{ for all } n \geq n_0$$

T is called the **period** of $\mathcal{M}$; the least n_0 is the **length of the initial segment**

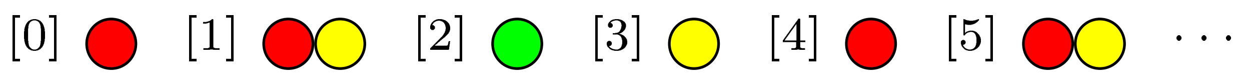
Theorem. *$\mathcal{M}_P$ is ultimately periodic.*

Crux: for every Statelog program P (which includes a database $\mathcal{D}$): $|\mathcal{D}| \leq |\mathcal{B}_P| < \infty$

$\Rightarrow$ there are at most $2^{|\mathcal{B}_P|}$ different database states (P does not invent new values). $\mathcal{M}_P$ has the *fixed window property* $\Rightarrow$ there must be a window repetition.

Example:

$[0]$ on(red)		$\leftarrow$	*% initial database state*
$[S{+}1]$	on(red),on(yellow)	$\leftarrow$	$[S]$ on(red), $\neg$ on(yellow)
$[S{+}1]$	on(green)	$\leftarrow$	$[S]$ on(red), on(yellow)
$[S{+}1]$	on(yellow)	$\leftarrow$	$[S]$ on(green)
$[S{+}1]$	on(red)	$\leftarrow$	$[S]$ on(yellow), $\neg$ on(red)



Theorem (Complexity).

- *Computing $\mathcal{M}_P[\$]$ is $\boldsymbol{PSPACE}$-complete*
- *There are programs whose period and length of the initial segment are exponential in the size of $\mathcal{D}$ (trick: encode a binary counter)*
- *If $\mathcal{M}_P$ is* inductive *(i.e., $\mathcal{M}_P[0] \subseteq \mathcal{M}_P[1] \subseteq \cdots$) then $\mathcal{M}_P[\$]$ always exists ($T = 1$) and can be computed in $\boldsymbol{PTIME}$*

- A program P **always/sometimes terminates** if $\mathcal{M}_{P \cup \mathcal{D}}$ terminates for **all/some** databases $\mathcal{D}$.

Theorem (Termination).

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

Every Statelog program P induces three natural mappings:

- $\Gamma^1_P : \mathcal{D} \mapsto \mathcal{M}_{P\cup\mathcal{D}}[1]$ *single transition* $[0] \leadsto [1]$
- $\Gamma^\$_P : \mathcal{D} \mapsto \mathcal{M}_{P\cup\mathcal{D}}[\$]$ *transaction:* $[0] \leadsto [\$]$
- $\Gamma^\infty_P : \mathcal{D} \mapsto \mathcal{M}_{P\cup\mathcal{D}}$ *database evolution* $[0] \leadsto [0]\,,[1]\,,[2]\,,\ldots$

Two Statelog programs P and Q are called

- **transition equivalent**, $P \equiv_1 Q$, if $\Gamma^1_P = \Gamma^1_Q$
- **transaction equivalent**, $P \equiv_\$ Q$, if $\Gamma^\$_P = \Gamma^\$_Q$
- **evolution equivalent**, $P \equiv_\infty Q$, if $\Gamma^\infty_P = \Gamma^\infty_Q$

Clearly, if $P \equiv_\infty Q$ then $P \equiv_1 Q$ and $P \equiv_\$ Q$

However, $P \equiv_\$ Q$ does not imply $P \equiv_\infty Q$

$\Rightarrow$ P and Q may always compute the same transaction, but in different ways:

$[S]$ tc(X,Y) $\leftarrow$ $[S]$ e(X,Y) $[S]$ tc(X,Y) $\leftarrow$ $[S]$ e(X,Z), $[S]$ tc(Z,Y)	$\equiv_\$$ $\not\equiv_\infty$	$[S{+}1]$ tc(X,Y) $\leftarrow$ $[S]$ e(X,Y) $[S{+}1]$ tc(X,Y) $\leftarrow$ $[S]$ e(X,Z), $[S]$ tc(Z,Y)

- *Statelog normal form* **NF-Statelog:** all rules have the form
  - $\boxed{[S{+}1]\ \ H \quad \leftarrow \quad [S]\ \ B \qquad \textit{\% (P) rule}}$ or
  - $\boxed{[S]\ \ H \quad \leftarrow \quad [S]\ \ B \qquad \textit{\% (L) rule}}$
- *Production rule normal form* **P-Statelog**: only (P) rules
  ($\widehat{=}$ NI-Datalog$^{\neg}$ and essentially I-Datalog$^{\neg}$, Datalog$^{\neg\neg}$)

Theorem (Normal Forms).

- *For every Statelog program there is an* ***evolution equivalent*** *NF-Statelog program.*
- *For every Statelog program there is a* ***transaction equivalent*** *P-Statelog program.*

Therefore,

- w.l.o.g. consider only NF-Statelog programs
- although P-Statelog programs have the same (transactional) expressive power, the intermediate state may differ significantly (cf. complement of transitive closure)
- using states explicitly, production rule semantics can be given a meaning within a declarative framework

Statelog ([LL94, LHL95, LML96])

- integrates
  - *production rules* which define state transitions, and
  - *deductive rules* which derive information within a state
- has a declarative semantics for negation: unique state-stratified model $\mathcal{M}_P$
- can be computed in the standard bottom-up way
- various variants have been investigated wrt. expressive power, complexity, termination

Some Expressiveness Results

Language	Equivalence wrt. Transitions	Equivalence wrt. Transactions
I-Statelog	S-Datalog	Fixpoint
Δ-Statelog	S-Datalog	Fixpoint
P-Statelog	$\exists$FO	While
XY-Datalog	FO	While
NF-Statelog	S-Datalog	While
Statelog	S-Datalog	While
W-Statelog	Fixpoint	While

Production Rules ([AV91]): special case of Statelog

Heraclitus ([GHJ$^+$93, GHJ96]): delta relations as *"first-class citizens"*

XY-Stratified Datalog ([Zan93, ZAO93]) $\approx$ variant of Statelog, where

- *X-rules* $\approx$ local rules (without recursion) $[S]\,H \ \leftarrow\ [S]\,B$
- *Y-rules* $\approx$ progressive rules $[S{+}1]\,H \ \leftarrow\ [S{+}1]\,B_1, [S]\,B_2$
- nondeterministic extensions (*choice* construct)

Datalog$_{1S}$, Templog ([Cho90, Bau89, BCW93])

- **query** language for **temporal databases** $\mathcal{D} \in (\mathcal{B}_P \rightarrow 2^{\mathbb{N}_0})$ (no updates)
- forward (progressive) **and backward** rules
- considers positive and stratified programs, but not state-stratified negation (essential for updates)
- **data expressiveness**[2] = set of expressible temporal databases. E.g. if $\mathcal{B}_P = \{q\}$ there are $2^{\mathbb{N}_0}$ temporal databases. From these, only the *ultimately periodic sets* are representable in Datalog$_{1S}$.
- **query expressiveness** = the class of expressible subsets of $(\mathcal{B}_P \rightarrow 2^{\mathbb{N}_0})$. For monadic Datalog$_{1S}$ (no data variables) = propositional Templog $\approx$ certain fragment of μTL [Bau95].

[2] not an issue for relational (finite one-state) databases

References

[AV91] S. Abiteboul and V. Vianu. Datalog Extensions for Database Queries and Updates. *Journal of Computer and System Sciences*, 43(1):62–124, 1991.

[Bau89] M. Baudinet. Temporal Logic Programming is Complete and Expressive. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 267–280, Austin, Texas, 1989.

[Bau95] M. Baudinet. On the Expressiveness of Temporal Logic Programming. *Information and Computation*, 117(2):157–180, 1995.

[BCW93] M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*. Benjamin/Cummings, 1993.

[Cho90] J. Chomicki. Polynomial Time Query Processing in Temporal Deductive Databases. In *Proc. ACM Symposium on Principles of Database Systems*, April 1990.

[GHJ+93] S. Ghandeharizadeh, R. Hull, D. Jacobs, J. Castillo, M. Escobar-Molano, S.-H. Lu, J. Luo, C. Tsang, and G. Zhou. On Implementing a Language for Specifying Active Database Execution Models. In *Proc. Intl. Conference on Very Large Data Bases*, 1993.

[GHJ96] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: Elevating Deltas to be First-Class Citizens in a Database Programming Language. *ACM Transactions on Database Systems*, 21(3):370–426, 1996.

[LHL95] B. Ludäscher, U. Hamann, and G. Lausen. A Logical Framework for Active Rules. In *Proc. 7th Intl. Conf. on Management of Data (COMAD)*, Pune, India, December 1995. Tata McGraw-Hill.

[LL94] G. Lausen and B. Ludäscher. Updates by Reasoning about States. In J. Eder and L. Kalinichenko, editors, *2nd Intl. East-West Database Workshop*, Workshops in Computing, Klagenfurt, Austria, 1994. Springer.

[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.

[Zan93] C. Zaniolo. A Unified Semantics for Active and Deductive Databases. In N. W. Paton and M. H. Williams, editors, *Proc. of the 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Workshops in Computing, Edinburgh, Scotland, 1993. Springer.

[ZAO93] C. Zaniolo, N. Arni, and K. Ong. Negation and Aggregates in Recursive Rules: the $\mathcal{LDL}++$ Approach. In S. Ceri, K. Tanaka, and S. Tsur, editors, *Proc. Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, number 760 in LNCS. Springer, 1993.