On Specifying Database Updates
Survey Talk on the
JLP article by Ray Reiter [Rei95]

Jens Bürger¹, Thomas Ruhroth¹ and Emanuel Sallinger²

¹TU Dortmund University, Dortmund, Germany
²Vienna University of Technology, Vienna, Austria

Research School FCCOD 2014, Bolzano, Italy
Overview

1. Situation Calculus
2. Database Transactions
3. Transaction Logs and Evaluation
4. Proving Properties of Database States
5. Extensions
6. Conclusion
Situation Calculus

**Situation calculus** is

- a logical language to represent **change**
- introduced by McCarthy [McC68]

A situation is

- “the complete state of the universe at an instance of time” (McCarthy and Hayes [MH69])
- the same as its history, i.e., the sequence of actions that has been performed since the initial situation (Reiter [Rei01])

For more background information, cf. Fangzhen Lin’s *Handbook of KR* article [Lin08]
Situation Calculus

**Situation calculus** is

- a logical language to represent **change**
- introduced by McCarthy [McC68]

A **situation** is

- “the complete state of the universe at an instance of time” (McCarthy and Hayes [MH69])
- the same as its history, i.e., the sequence of actions that has been performed since the initial situation (Reiter [Rei01])

For more background information, cf. Fangzhen Lin’s *Handbook of KR* article [Lin08]
Situation Calculus

Situation calculus is

- a logical language to represent **change**
- introduced by McCarthy [McC68]

A situation is

- “the complete state of the universe at an instance of time” (McCarthy and Hayes [MH69])
- the same as its history, i.e., the sequence of actions that has been performed since the initial situation (Reiter [Rei01])

For more background information, cf. Fangzhen Lin’s *Handbook of KR* article [Lin08]
Situation Calculus

A logical language over a vocabulary of

- **fluent**s: relation symbols like $\text{broken}(x, s)$ where the last argument always refers to the situation
- **actions**: function symbols like $\text{repair}(r, x)$
- **atemporals**: relation symbols like $\text{heavy}(x)$ that hold regardless of the situation

The vocabulary also includes the special symbols:

- the predicate $\text{Poss}(\text{action}, \text{situation})$ indicates that an action is possible in a certain situation
- the function $\text{do}(\text{action}, \text{situation})$ describes the resulting situation
Situation Calculus

A logical language over a vocabulary of

- **fluent**s: relation symbols like \( \text{broken}(x, s) \)
  where the last argument always refers to the situation

- **actions**: function symbols like \( \text{repair}(r, x) \)

- **atemporals**: relation symbols like \( \text{heavy}(x) \)
  that hold regardless of the situation

The vocabulary also includes the special symbols:

- the predicate \( \text{Poss}(action, situation) \)
  indicates that an action is possible in a certain situation

- the function \( \text{do}(action, situation) \)
  describes the resulting situation
Situation Calculus

A logical language over a vocabulary of

- **fluenets**: relation symbols like $\text{broken}(x, s)$ where the last argument always refers to the situation
- **actions**: function symbols like $\text{repair}(r, x)$
- **atemporals**: relation symbols like $\text{heavy}(x)$ that hold regardless of the situation

The vocabulary also includes the special symbols:

- the predicate $\text{Poss}(\text{action}, \text{situation})$ indicates that an action is possible in a certain situation
- the function $\text{do}(\text{action}, \text{situation})$ describes the resulting situation
Situation Calculus

A logical language over a vocabulary of

- **fluents**: relation symbols like $\text{broken}(x, s)$
  where the last argument always refers to the situation
- **actions**: function symbols like $\text{repair}(r, x)$
- **atemporals**: relation symbols like $\text{heavy}(x)$
  that hold regardless of the situation

The vocabulary also includes the special symbols:

- the predicate $\text{Poss}(\text{action}, \text{situation})$
  indicates that an action is **possible** in a certain situation
- the function $\text{do}(\text{action}, \text{situation})$
  describes the resulting situation
A logical language over a vocabulary of

- **fluențes**: relation symbols like $\text{broken}(x, s)$
  where the last argument always refers to the situation
- **actions**: function symbols like $\text{repair}(r, x)$
- **atemporals**: relation symbols like $\text{heavy}(x)$
  that hold regardless of the situation

The vocabulary also includes the special symbols:

- the predicate $\text{Poss}(\text{action}, \text{situation})$
  indicates that an action is possible in a certain situation
- the function $\text{do}(\text{action}, \text{situation})$
  describes the resulting situation
Precondition axioms:

- \( \text{broken}(x, s) \land \text{hasGlue}(r, s) \rightarrow \text{Poss}(\text{repair}(r, x), s) \)
- \( [\forall z \neg \text{holding}(r, z, s)] \land \neg \text{heavy}(x) \land \text{nextTo}(r, x, s) \rightarrow \text{Poss}(\text{repair}(r, x), s) \)

Effect axioms:

- \( \text{Poss}(\text{repair}(r, x), s) \rightarrow \neg \text{broken}(x, \text{do}(\text{repair}(r, x), s)) \)
- \( \text{Poss}(\text{drop}(r, x), s) \land \text{fragile}(x) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s)) \)
Precondition axioms:

- $\text{broken}(x, s) \land \text{hasGlue}(r, s) \rightarrow \text{Poss}(\text{repair}(r, x), s)$
- $[\forall z \neg \text{holding}(r, z, s)] \land \neg \text{heavy}(x) \land 
\text{nextTo}(r, x, s) \rightarrow \text{Poss}(\text{repair}(r, x), s)$

Effect axioms:

- $\text{Poss}(\text{repair}(r, x), s) \rightarrow \neg \text{broken}(x, \text{do}(\text{repair}(r, x), s))$
- $\text{Poss}(\text{drop}(r, x), s) \land 
\text{fragile}(x) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s))$
Precondition axioms:

- \( \text{broken}(x, s) \land \text{hasGlue}(r, s) \rightarrow \text{Poss}(\text{repair}(r, x), s) \)
- \( [\forall z \neg \text{holding}(r, z, s)] \land \neg \text{heavy}(x) \land \text{nextTo}(r, x, s) \rightarrow \text{Poss}(\text{repair}(r, x), s) \)

Effect axioms:

- \( \text{Poss}(\text{repair}(r, x), s) \rightarrow \neg \text{broken}(x, \text{do}(\text{repair}(r, x), s)) \)
- \( \text{Poss}(\text{drop}(r, x), s) \land \text{fragile}(x) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s)) \)
Precondition axioms:

- $\text{broken}(x, s) \land \text{hasGlue}(r, s) \rightarrow \text{Poss}(\text{repair}(r, x), s)$
- $[\forall z \neg \text{holding}(r, z, s)] \land \neg \text{heavy}(x) \land \text{nextTo}(r, x, s) \rightarrow \text{Poss}(\text{repair}(r, x), s)$

Effect axioms:

- $\text{Poss}(\text{repair}(r, x), s) \rightarrow \neg \text{broken}(x, \text{do}(\text{repair}(r, x), s))$
- $\text{Poss}(\text{drop}(r, x), s) \land \text{fragile}(x) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s))$
The Frame Problem

The frame problem is

- one of the most famous AI problems
- “normally, only relatively few actions [...] will affect the truth value of a given fluent”

Frame axioms:

1. \( \text{Poss}(\text{drop}(r, x), s) \land \text{color}(y, c, s) \rightarrow \text{color}(y, c, \text{do}(\text{drop}(r, x), s)) \)
2. \( \text{Poss}(\text{drop}(r, x), s) \land \neg \text{broken}(y, s) \land 
   [y \neq x \lor \neg \text{fragile}(y)] \rightarrow \neg \text{broken}(y, \text{do}(\text{drop}(r, x), s)) \)
The Frame Problem

The frame problem is

- one of the most famous AI problems
- “normally, only relatively few actions [...] will affect the truth value of a given fluent”

Frame axioms:

- \( \text{Poss}(\text{drop}(r, x), s) \land \text{color}(y, c, s) \rightarrow \text{color}(y, c, \text{do(\text{drop}(r, x), s)}) \)
- \( \text{Poss}(\text{drop}(r, x), s) \land \neg \text{broken}(y, s) \land \\
[y \neq x \lor \neg \text{fragile}(y)] \rightarrow \neg \text{broken}(y, \text{do(\text{drop}(r, x), s)}) \)
The Frame Problem

The frame problem is

- one of the most famous AI problems
- “normally, only relatively few actions [...] will affect the truth value of a given fluent”

Frame axioms:

- $\text{Poss}(\text{drop}(r, x), s) \land \color(y, c, s) \rightarrow \color(y, c, \text{do}(\text{drop}(r, x), s))$
- $\text{Poss}(\text{drop}(r, x), s) \land \neg \text{broken}(y, s) \land \left[ y \neq x \lor \neg \text{fragile}(y) \right] \rightarrow \neg \text{broken}(y, \text{do}(\text{drop}(r, x), s))$
Some database relations are modeled as **fluents**:

- \textit{enrolled}(student, course, s)
- \textit{grade}(student, course, grade, s)

Some as atemporals:

- \textit{prereq}(prerequisite, course)
Modeling Databases

Some database relations are modeled as **fluents**:

- \( \text{enrolled}(\text{student}, \text{course}, s) \)
- \( \text{grade}(\text{student}, \text{course}, \text{grade}, s) \)

Some as **atemporals**:

- \( \text{prereq}(\text{prerequisite}, \text{course}) \)
Transactions (changes to the database) are modeled as actions:

- *register*(student, course)
- *change*(student, course, grade)
- *drop*(student, course)
Modeling Preconditions

Most transactions have particular preconditions:

- \( \text{Poss}(\text{drop}(st, c), s) \leftrightarrow \text{enrolled}(st, c, s) \)
- \( \text{Poss}(\text{register}(st, c), s) \leftrightarrow \)
  \[ \forall p \text{ prereq}(p, c) \rightarrow \exists g \text{ grade}(st, p, g, s) \land g \geq 50 \]
- \( \text{Poss}(\text{change}(st, c, g), s) \leftrightarrow \)
  \[ \exists g' \text{ grade}(st, c, g', s) \land g' \neq g \]

Observe the common syntactic form of these preconditions!
Modeling Preconditions

Most transactions have particular preconditions:

- \( \textbf{Poss}(\text{drop}(st, c), s) \leftrightarrow \textbf{enrolled}(st, c, s) \)
- \( \textbf{Poss}(\text{register}(st, c), s) \leftrightarrow \)
  \[ [\forall p \text{ prereq}(p, c)] \rightarrow [\exists g \text{ grade}(st, p, g, s) \land g \geq 50] \]
- \( \textbf{Poss}(\text{change}(st, c, g), s) \leftrightarrow \)
  \[ [\exists g' \text{ grade}(st, c, g', s) \land g' \neq g] \]

Observe the common syntactic form of these preconditions!
Modeling Preconditions

Most transactions have particular preconditions:

- **Poss**(drop(st, c), s) ↔ enrolled(st, c, s)
- **Poss**(register(st, c), s) ↔
  \[ \forall p \ \text{prereq}(p, c) \rightarrow \exists g \ \text{grade}(st, p, g, s) \land g \geq 50 \]
- **Poss**(change(st, c, g), s) ↔
  \[ \exists g' \ \text{grade}(st, c, g', s) \land g' \neq g \]

*Observe the common syntactic form of these preconditions!*
Modeling Effects

The most important and usually most complex parts are the effects of transactions:

- **Poss**$(a, s) \rightarrow \left[ \text{enrolled}(st, c, do(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c)) \right]$

- **Poss**$(a, s) \rightarrow \left[ \text{grade}(st, c, g, do(a, s)) \leftrightarrow a = \text{change}(st, c, g) \lor (\text{grade}(st, c, g, s) \land \forall g' g' \neq g \rightarrow a \neq \text{change}(st, c, g')) \right]$

Observe the syntactic form and in particular the (implicit) universal quantification over transactions!
Modeling Effects

The most important and usually most complex parts are the effects of transactions:

- **Poss**\((a, s) \rightarrow \lbrack \text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c))\)\]

- **Poss**\((a, s) \rightarrow \lbrack \text{grade}(st, c, g, \text{do}(a, s)) \leftrightarrow a = \text{change}(st, c, g) \lor (\text{grade}(st, c, g, s) \land \forall g' \ g' \neq g \rightarrow a \neq \text{change}(st, c, g'))\)\]

Observe the syntactic form and in particular the (implicit) universal quantification over transactions!
The database relation \textit{enrolled} can only be affected by transactions \textit{register} or \textit{drop}.
The Frame Problem Revisited

- \( \text{Poss}(a, s) \rightarrow [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c))] \)

implies

- \( \text{Poss}(a, s) \land a \neq \text{register}(st, c) \land a \neq \text{drop}(st, c) \rightarrow [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow \text{enrolled}(st, c, s)] \)

“The database relation \text{enrolled} can only be affected by transactions \text{register} or \text{drop}.”
The Frame Problem Revisited

\[ \text{Poss}(a, s) \rightarrow [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c))] \]

implies

\[ \text{Poss}(a, s) \land \\
\quad a \neq \text{register}(st, c) \land a \neq \text{drop}(st, c) \rightarrow \\
\quad [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow \text{enrolled}(st, c, s)] \]

“The database relation \textit{enrolled} can only be affected by transactions \textit{register} or \textit{drop}.”
The Frame Problem Revisited

\[ \text{Poss}(a, s) \rightarrow [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c))] \]

Succinct representation of the frame axioms is possible because:

- quantification over all transactions
- the assumption that “few” transactions affect a particular database relation
What if we want to know

“Is John enrolled in any course after transaction sequence
\[ \text{\textit{drop}(John, C100), register}(Mary, C100) \]
from initial state \( S_0 \)?”

We need to evaluate over our database the formula

\[
\exists c \ \text{enrolled}(John, c, \\
do(\text{register}(Mary, C100), \\
do(\text{drop}(John, C100), S_0)))
\]

This is called the temporal projection problem.
What if we want to know

“Is John enrolled in any course after transaction sequence
\(\text{drop}(John, C100), \text{register}(Mary, C100)\)
from initial state \(S_0\)?”

We need to evaluate over our database the formula

\[\exists c \text{ enrolled}(John, c,\]
\[\text{do}(\text{register}(Mary, C100),\]
\[\text{do}(\text{drop}(John, C100), S_0)))\]

This is called the \textit{temporal projection problem}.
The situation calculus used is

- a first-order language
- with equality and $<$
- that is many-sorted (actions, situations)

**But** we later need one second-order feature, namely

- quantification over situations
Axiomatizing Transactions

The situation calculus used is

- a first-order language
- with equality and $<$
- that is many-sorted (actions, situations)

But we later need one second-order feature, namely

- quantification over situations
Axiomatizing Transactions

Unique name assumption for
- transactions (i.e. actions)
- states (i.e. situations)

In particular, for transactions it is enforced that

\[ t(x_1, \ldots, x_n) = t'(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \land \ldots \land x_n = y_n \]

This actually means that

Two states are equal if they have the same history, it is not enough for them to have equal values for all fluents.
Axiomatizing Transactions

Unique name assumption for

- transactions (i.e. actions)
- states (i.e. situations)

In particular, for transactions it is enforced that

\[ t(x_1, \ldots, x_n) = t'(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \land \ldots \land x_n = y_n \]

This actually means that

Two states are equal if they have the same history, it is \textit{not} enough for them to have equal values for all fluents.
Axiomatizing Transactions

Unique name assumption for

- transactions (i.e. actions)
- states (i.e. situations)

In particular, for transactions it is enforced that

$$t(x_1, \ldots, x_n) = t'(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \land \ldots \land x_n = y_n$$

This actually means that

Two states are equal if they have the same history, it is not enough for them to have equal values for all fluents.
Simple Formulas

Recall the example:

- $$\text{Poss}(\text{drop}(st, c), s) \leftrightarrow \text{enrolled}(st, c, s)$$
- $$\text{Poss}(\text{register}(st, c), s) \leftrightarrow \left[ \forall p \, \text{prereq}(p, c) \right] \to \left[ \exists g \, \text{grade}(st, p, g, s) \land g \geq 50 \right]$$
- $$\text{Poss}(\text{change}(st, c, g), s) \leftrightarrow \left[ \exists g' \, \text{grade}(st, c, g', s) \land g' \neq g \right]$$

A simple formula is a first-order formula that

- does not contain $$\text{Poss}$$ or $$\text{do}$$
- does not quantify over states
Simple Formulas

Recall the example:

- \( \text{Poss}(\text{drop}(st, c), s) \leftrightarrow \text{enrolled}(st, c, s) \)
- \( \text{Poss}(\text{register}(st, c), s) \leftrightarrow \) 
  \[ \forall p \text{ prereq}(p, c) \rightarrow [\exists g \text{ grade}(st, p, g, s) \land g \geq 50] \]
- \( \text{Poss}(\text{change}(st, c, g), s) \leftrightarrow \) 
  \[ [\exists g' \text{ grade}(st, c, g', s) \land g' \neq g] \]

A **simple formula** is a first-order formula that

- does not contain \( \text{Poss} \) or \( \text{do} \)
- does not quantify over states
Transaction Precondition Axioms

Recall the example:

- \( \text{Poss} \left( \text{drop}(st, c), s \right) \leftrightarrow \text{enrolled}(st, c, s) \)
- \( \text{Poss} \left( \text{register}(st, c), s \right) \leftrightarrow \)
  \[ [\forall p \text{prereq}(p, c)] \rightarrow [\exists g \text{grade}(st, p, g, s) \land g \geq 50] \]
- \( \text{Poss} \left( \text{change}(st, c, g), s \right) \leftrightarrow \)
  \[ [\exists g' \text{grade}(st, c, g', s) \land g' \neq g] \]

A transaction precondition axiom has the form

\[ \forall \vec{x} \forall s \text{Poss}(\text{transaction}(x_1, \ldots, x_n), s) \leftrightarrow \Pi_{\text{transaction}} \]
Transaction Precondition Axioms

Recall the example:

- \( \text{Poss}(\text{drop}(st, c), s) \leftrightarrow \text{enrolled}(st, c, s) \)
- \( \text{Poss}(\text{register}(st, c), s) \leftrightarrow [\forall p \text{ prereq}(p, c)] \rightarrow [\exists g \text{ grade}(st, p, g, s) \land g \geq 50] \)
- \( \text{Poss}(\text{change}(st, c, g), s) \leftrightarrow [\exists g' \text{ grade}(st, c, g', s) \land g' \neq g] \)

A transaction precondition axiom has the form

\[ \forall \vec{x} \forall s \text{ Poss}(\text{transaction}(x_1, \ldots, x_n), s) \leftrightarrow \Pi_{\text{transaction}} \]
Successor State Axioms

Recall the example:

- **Poss** \((a, s) \rightarrow \left[ \text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow \begin{align*} & a = \text{register}(st, c) \lor \\ & (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c)) \end{align*} \right] \)

- **Poss** \((a, s) \rightarrow \left[ \text{grade}(st, c, g, \text{do}(a, s)) \leftrightarrow \begin{align*} & a = \text{change}(st, c, g) \lor \\ & (\text{grade}(st, c, g, s) \land [\forall g' g' \neq g \rightarrow a \neq \text{change}(st, c, g')]) \end{align*} \right] \)

A successor state axiom has the form

\[ \forall a \forall s \ \text{Poss}(a, s) \rightarrow \forall \vec{x} \ \text{fluent}(x_1, \ldots, x_n, \text{do}(a, s)) \leftrightarrow \Phi_{\text{fluent}} \]
Successor State Axioms

Recall the example:

- \( \text{Poss}(a, s) \rightarrow [\text{enrolled}(st, c, \text{do}(a, s)) \leftrightarrow a = \text{register}(st, c) \lor (\text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c))] \)
- \( \text{Poss}(a, s) \rightarrow [\text{grade}(st, c, g, \text{do}(a, s)) \leftrightarrow a = \text{change}(st, c, g) \lor (\text{grade}(st, c, g, s) \land [\forall g' g' \neq g \rightarrow a \neq \text{change}(st, c, g')])] \)

A successor state axiom has the form

\[ \forall a \forall s \text{Poss}(a, s) \rightarrow \forall \vec{x} \text{fluent}(x_1, \ldots, x_n, \text{do}(a, s)) \leftrightarrow \Phi_{\text{fluent}} \]
The Frame Problem Solved

Key to Reiter’s solution to the Frame Problem are successor state axioms like

\[
\text{Poss}(a, s) \rightarrow [\text{grade}(st, c, g, \text{do}(a, s)) \leftrightarrow a = \text{change}(st, c, g) \lor (\text{grade}(st, c, g, s) \land [\forall g' \ g' \neq g \rightarrow a \neq \text{change}(st, c, g')])]
\]

A tuple is contained in the database if and only if

- it is added by a transaction
- it was there and is not deleted by a transaction
Key to Reiter’s solution to the Frame Problem are successor state axioms like

\[ \text{Poss}(a, s) \rightarrow [\text{grade}(st, c, g, \text{do}(a, s)) \leftrightarrow \\
a = \text{change}(st, c, g) \lor \\
(\text{grade}(st, c, g, s) \land [\forall g' \ g' \neq g \rightarrow a \neq \text{change}(st, c, g')])] \]

A tuple is contained in the database if and only if

- it is added by a transaction
- it was there and is not deleted by a transaction
In Database applications,

- a *log* is a sequence of update transactions
- queries are processed wrt. the log
- transactions (esp. here) are *virtual*

**Questions to be addressed**

Given: Query $Q$, transaction sequence $\tau_1, \ldots, \tau_n$

- Is $\tau_1, \ldots, \tau_n$ a legal sequence?
- What is the answer to $Q$, wrt. $S_0$?
In Database applications,
- a log is a sequence of update transactions
- queries are processed wrt. the log
- transactions (esp. here) are virtual

Questions to be addressed

Given: Query $Q$, transaction sequence $\tau_1, \ldots, \tau_n$
- Is $\tau_1, \ldots, \tau_n$ a legal sequence?
- What is the answer to $Q$, wrt. $S_0$?
Legal Transaction Sequences

- Illegal transaction sequences fairly exist:

Example

- \textit{drop}(Sue, C100), \textit{change}(Bill, C100, 60)

Is false, if e.g. \textbf{Poss}(\textit{drop}(Sue, C100), S_0)) is false.

Transaction sequence is legal iff:

- beginning in state \( S_0 \)
- each transaction in the sequence is possible and results from the preceding one

Ordering Relation \(<\) on states

\[(\forall s) \neg s < S_0 \quad (1)\]

\[(\forall a, s, s'). s < \textbf{do}(a, s') \leftrightarrow \textbf{Poss}(a, s') \land s \leq s' \quad (2)\]
Legal Transaction Sequences

- Illegal transaction sequences fairly exist:

Example

- \textit{drop}(Sue, C100), \textit{change}(Bill, C100, 60)
- Is false, if e.g. \textbf{Poss}(\textit{drop}(Sue, C100), S_0)) is

Transaction sequence is legal iff:

- beginning in state $S_0$
- each transaction in the sequence is possible and results from the preceeding one

Ordering Relation $<$ on states

\begin{align*}
(\forall s) \neg s < S_0 \quad & (1) \\
(\forall a, s, s'). s < \text{do}(a, s') & \iff \text{Poss}(a, s') \land s \leq s' \quad & (2)
\end{align*}
Legal Transaction Sequences

- Illegal transaction sequences fairly exist:

Example

- `drop(Sue, C100)`, `change(Bill, C100, 60)`
- Is false, if e.g. `Poss(drop(Sue, C100), S₀)` is

Transaction sequence is legal iff:
- beginning in state `S₀`
- each transaction in the sequence is possible and results from the preceding one

Ordering Relation `<` on states

\[
(\forall s) \neg s < S₀ \tag{1}
\]
\[
(\forall a, s, s'). s < \text{do}(a, s') \iff \text{Poss}(a, s') \land s \leq s' \tag{2}
\]
Legal Transaction Sequences

Induction Principle

- Common induction principle to be used later on:

$$(\forall P). P(S_0) \land (\forall a, s)[P(s) \rightarrow P(\text{do}(a, s))] \rightarrow (\forall s)P(s). \quad (3)$$

- Compare with the induction axiom for natural numbers:

$$(\forall P). P(0) \land (\forall x)[P(x) \rightarrow P(\text{succ}(x))] \rightarrow (\forall x)P(x).$$
Legal Transaction Sequences

Induction Principle

- Common induction principle to be used later on:

\[
(\forall P).P(S_0) \land (\forall a, s)[P(s) \rightarrow P(\text{do}(a, s))] \rightarrow (\forall s)P(s). 
\] (3)

- Compare with the induction axiom for natural numbers:

\[
(\forall P).P(0) \land (\forall x)[P(x) \rightarrow P(\text{succ}(x))] \rightarrow (\forall x)P(x). 
\]
Legal Transaction Sequences

Definition of database

- Given: sequence of transaction terms $\tau_1, \ldots, \tau_n$
- The sequence is legal iff

$$D \models S_0 \leq \text{do}([\tau_1, \ldots, \tau_n])$$

while Database $D$ is formalized as:

$$D = \Sigma \cup D_{ss} \cup D_{tp} \cup D_{uns} \cup D_{unt} \cup D_{S_0}$$

- $\Sigma$: set of the three state axioms
- $D_{ss}$: set of successor state axioms
- $D_{tp}$: set of transaction precondition axioms
- $D_{uns}$: set of unique names axioms for states
- $D_{unt}$: set of unique names axioms for transactions
- $D_{S_0}$: set of FO sentences with only $S_0$ referenced
Legal Transaction Sequences

Definition of database

- Given: sequence of transaction terms $\tau_1, \ldots, \tau_n$
- The sequence is legal iff

\[ \mathcal{D} \models S_0 \leq \text{do}(\tau_1, \ldots, \tau_n) \]

while Database $\mathcal{D}$ is formalized as:

\[ \mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{tp} \cup \mathcal{D}_{uns} \cup \mathcal{D}_{unt} \cup \mathcal{D}_{S_0} \]

- $\Sigma$: set of the three state axioms
- $\mathcal{D}_{ss}$: set of successor state axioms
- $\mathcal{D}_{tp}$: set of transaction precondition axioms
- $\mathcal{D}_{uns}$: set of unique names axioms for states
- $\mathcal{D}_{unt}$: set of unique names axioms for transactions
- $\mathcal{D}_{S_0}$: set of FO sentences with only $S_0$ referenced

$\rightsquigarrow$ initial database
Regression operator $\mathcal{R}$

- *unfolding* operation
- reduce complexity of ground terms
- application may lead to formula with $S_0$ as only state term
- $\Rightarrow$ reduced complexity in theorem proving

Usage:
- defined recursively using formula substitution
- recursively substitutes parts of a formular into their successor state axioms
- reduces depth of nesting function symbol $\text{do}$ in formulae
- $\mathcal{R}^n$ lets $\mathcal{R}$ be applied in a nested way:
  - For $n=1,2,\ldots$:
    - $\mathcal{R}^n[G] = \mathcal{R}[\mathcal{R}^{n-1}[G]]$ aso.

$^1$terms not mentioning any variable
Legal Transaction Sequences
Regression Operator

Regression operator $\mathcal{R}$

- *unfolding* operation
- reduce complexity of ground terms¹
- application may lead to formula with $S_0$ as only state term
- $\rightsquigarrow$ reduced complexity in theorem proving

Usage:

- defined recursively using formula substitution
- recursively substitutes parts of a formular into their successor state axioms
- reduces depth of nesting function symbol *do* in formulae
- $\mathcal{R}^n$ lets $\mathcal{R}$ be applied in a nested way:

  - For $n=1,2,\ldots$:
  
  \[ \mathcal{R}^n[G] = \mathcal{R}[\mathcal{R}^{n-1}[G]] \] aso.

¹terms not mentioning any variable
Theorem [Rei95]:

The sequence $\tau_1, \ldots, \tau_n$ [...] of sort transaction is legal wrt. $\mathcal{D}$ iff

$$\mathcal{D}_{unt} \cup \mathcal{D}_{S_0} \models \bigwedge_{i=1}^{n} \mathcal{R}^{i-1}[\text{precond}(\tau_i, \text{do}([\tau_1, \ldots, \tau_{i-1}], S_0))].$$

$\text{precond}(\tau, s)$ specifies circumstances under which ground transaction $\tau$ is possible in state $s$. 
Legal Transaction Sequences

Example: Legality Testing

Consider following transaction sequence:

Example

\textit{register}(Bill, C100), \textit{drop}(Bill, C100), \textit{drop}(Bill, C100)

\begin{align*}
\mathcal{R}^0[\text{precond}(\text{register}(Bill, C100), S_0)] \land \\
\mathcal{R}^1[\text{precond}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0))] \land \\
\mathcal{R}^2[\text{precond}(\text{drop}(Bill, C100), \text{do}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0)))]
\end{align*}
Consider following transaction sequence:

Example

\textit{register}(Bill, C100), \textit{drop}(Bill, C100), \textit{drop}(Bill, C100)

\[\mathcal{R}^0[\text{precond}(\text{register}(Bill, C100), S_0)] \land \mathcal{R}^1[\text{precond}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0))] \land \mathcal{R}^2[\text{precond}(\text{drop}(Bill, C100), \text{do}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0))))]\]
which is

\[ R^0[(\forall p).\text{prerequ}(p, C100) \rightarrow (\exists g).\text{grade}(Bill, p, g, S_0) \land g \geq 50] \land \\
R^1[\text{enrolled}(Bill, C100, \text{do}(\text{register}(Bill, C100), S_0))] \land \\
R^2[\text{enrolled}(Bill, C100), \text{do}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0)))] \]

which leads to

\{ (\forall p).\text{prerequ}(p, C100) \rightarrow (\exists g).\text{grade}(Bill, p, g, S_0) \land g \geq 50 \} \land \\
true \land \\
false
Legal Transaction Sequences

Example: Legality Testing (cont’d)

which is

$$\mathcal{R}^0[\forall p.\text{prerequ}(p, C100) \rightarrow (\exists g).\text{grade}(Bill, p, g, S_0) \land g \geq 50]\land$$

$$\mathcal{R}^1[\text{enrolled}(Bill, C100, \text{do}(\text{register}(Bill, C100), S_0))]\land$$

$$\mathcal{R}^2[\text{enrolled}(Bill, C100), \text{do}(\text{drop}(Bill, C100), \text{do}(\text{register}(Bill, C100), S_0))]]$$

which leads to

$$\{(\forall p).\text{prerequ}(p, C100) \rightarrow (\exists g).\text{grade}(Bill, p, g, S_0) \land g \geq 50\}\land$$

true\land false$$

Bürger, Ruhroth and Sallinger ()

On Specifying Database Updates

FCCOD ’2014 27 / 44
Query Evaluation

- Given: Sequence $\tau_1, \ldots, \tau_n$ of transaction terms
- Query $Q(s)$

What is the answer to $Q$ in the state that results by applying $\tau_1, \ldots, \tau_i$ beginning with database in state $S_0$?

Formally:

$$\mathcal{D} \models Q(\text{do}(\tau_1, \ldots, \tau_n, S_0))$$

Reiter’s result

Given a legal transaction sequence $\tau_1, \ldots, \tau_n$,

$$\mathcal{D} \models Q(\text{do}(\tau_1, \ldots, \tau_n, S_0))$$

iff

$$\mathcal{D}_{\text{unt}} \cup \mathcal{D}_{S_0} \models \mathcal{R}^n[Q(\text{do}[\tau_1, \ldots, \tau_n], S_0))]$$
Query Evaluation

- Given: Sequence $\tau_1, \ldots, \tau_n$ of transaction terms
- Query $Q(s)$

What is the answer to $Q$ in the state that results by applying $\tau_1, \ldots, \tau_i$ beginning with database in state $S_0$?

Formally:

$$\mathcal{D} \models Q(\text{do}([\tau_1, \ldots, \tau_n], S_0))$$

Reiter's result

Given a legal transaction sequence $\tau_1, \ldots, \tau_n$,

$$\mathcal{D} \models Q(\text{do}([\tau_1, \ldots, \tau_n], S_0))$$

iff

$$\mathcal{D}_{\text{unt}} \cup \mathcal{D}_{S_0} \models \mathcal{R}^n[Q(\text{do}[\tau_1, \ldots, \tau_n], S_0))]$$
Query Evaluation

- Given: Sequence $\tau_1, \ldots, \tau_n$ of transaction terms
- Query $Q(s)$

What is the answer to $Q$ in the state that results by applying $\tau_1, \ldots, \tau_i$ beginning with database in state $S_0$?

Formally:

$$D \models Q(\text{do}([\tau_1, \ldots, \tau_n], S_0))$$

Reiter’s result

Given a legal transaction sequence $\tau_1, \ldots, \tau_n$,

$$D \models Q(\text{do}([\tau_1, \ldots, \tau_n], S_0))$$

iff

$$\mathcal{D}_{\text{unt}} \cup \mathcal{D}_{S_0} \models \mathcal{R}^n[Q(\text{do}[\tau_1, \ldots, \tau_n], S_0))]$$
Query Evaluation
Example

- Given:
  \[ T = \text{change}(Bill, C100, 60), \text{register}(Sue, C200), \text{drop}(Bill, C100) \]

- Query:
  \[
  (\exists st).\text{enrolled}(st, C200, \text{do}(T, S_0)) \land \\
  \lnot \text{enrolled}(st, C100, \text{do}(T, S_0)) \land \\
  (\exists g).\text{grade}(st, C200, g, \text{do}(T, S_0)) \land g \geq 50
  \]

- \( \rightsquigarrow \mathcal{R}^3 \) needs to be computed.

- Applying some simplifications (and assume \( \mathcal{D}_{S_0} \models C100 \neq C200 \)):
  \[
  (\exists st).[st = Sue \lor \text{enrolled}(st, C200, S_0)] \land \\
  [st = Bill \lor \lnot \text{enrolled}(st, C100, S_0)] \land \\
  [(\exists g).\text{grade}(st, C200, g, S_0) \land g \geq 50]
  \]
Query Evaluation

Example

Given:
\[ T = change(Bill, C100, 60), \text{register}(Sue, C200), \text{drop}(Bill, C100) \]

Query:
\[
(\exists st).enrolled(st, C200, do(T, S_0)) \land \\
\neg \text{enrolled}(st, C100, do(T, S_0)) \land \\
(\exists g).\text{grade}(st, C200, g, do(T, S_0)) \land g \geq 50
\]

\[ \rightsquigarrow R^3 \] needs to be computed.

Applying some simplifications (and assume \( D_{S_0} \models C100 \neq C200 \)):
\[
(\exists st).[st = Sue \lor enrolled(st, C200, S_0)] \land \\
[st = Bill \lor \neg enrolled(st, C100, S_0)] \land \\
[(\exists g).\text{grade}(st, C200, g, S_0) \land g \geq 50]
\]
Recall analogy between natural numbers and database updates:

- let $S_0$ be identified with 0 and $\textbf{do}(\text{Add}1, s)$ as the successor of the natural number $s$.

Reiter introduces two induction principles:

- $IP_{S_0 \leq s}$
  - (a property holds \textit{all the time})

- $IP_{S_0 \leq s \land s \leq s'}$
  - (a property holds \textit{between} two states $s, s'$)

$\Rightarrow$ Can be used to prove

- functional dependencies (when using \textit{grade}, all the other grades remain unchanged)

- dynamic integrity constraints (dynamically checking if salary of an employee ever decreases)
Recall analogy between natural numbers and database updates:

- let $S_0$ be identified with $0$ and $\text{do}(\text{Add} 1, s)$ as the successor of the natural number $s$

Reiter introduces two induction principles:

- $IP_{S_0 \leq s}$ (a property holds *all the time*)
- $IP_{S_0 \leq s \land s \leq s'}$ (a property holds *between* two states $s, s'$)

$\Rightarrow$ Can be used to prove

- functional dependencies (when using *grade*, all the other grades remain unchanged)
- dynamic integrity constraints (dynamically checking if salary of an employee ever decreases)
Extensions

- Transaction Logs and Historical Queries
- Complexity of Query Evaluation
- Actualizing Transactions
- Updates in the Logic Programming Context
- Views
- State Constraints and the Ramification and Qualification Problems

Focus
Extensions

- Transaction Logs and Historical Queries
- Complexity of Query Evaluation
- Actualizing Transactions
- Updates in the Logic Programming Context
- Views
- State Constraints and the Ramification and Qualification Problems

Focus
Transaction Logs and Historical Queries
Problem of Historical Queries

Action Example: Has some action happened in the history?
Has Mary dropped the course C100?\
\textit{drop}(\textit{Mary}, \textit{C100})

Property Example: Has some action happened in the history?
Has Sue always worked in Department 13?\
\textit{amp}(\textit{Sue}, 13, s)

Action Example: Has some action happened in a part of the history?
Has Mary dropped the course C100 between situation $s$ and $s'$?\
\textit{drop}(\textit{Mary}, \textit{C100})
Formalization using $<\ operator$

**Specific Point in History**

$(\exists s). S_0 \leq s \land s \leq s' \land someprop(s)$

$(\exists s). S_0 \leq s \land s \leq do(T, S_0) \land someprop(s)$

**Whole History**

$(\forall s). S_0 \leq s \land s \leq s' \rightarrow someprop(s)$

$(\forall s). S_0 \leq s \land s \leq do(T, S_0) \rightarrow someprop(s)$

**Part of History**

$(occurs \ – \ between(a, s, s')) \triangleq (\exists s''). s < do(a, s'') < s'$
Examples formalized

Has Mary dropped the course C100?

\[(\exists s, s'). S_0 \leq s \land s \leq \text{do}(T, S_0) \land s = \text{do}(	ext{drop}(\text{Mary}, \text{C100}), s')\]

Has Sue always worked in Department 13?

\[(\forall s). S_0 \leq s \land s \leq \text{do}(T, S_0) \rightarrow \text{emp}(Sue, 13, s)\]

Has Mary dropped the course C100 between two situation s and s'?

\[(\text{occurs} \rightarrow \text{between}(	ext{drop}(\text{Mary}, \text{C100}), s, s'))\]
Performing Queries - Idea

Transform into “Action-Form”

\[
\begin{align*}
\text{\textit{emp}}(Sue, 13, S_0)\land \\
\neg\text{\textit{occurs}} \rightarrow \text{\textit{between}}(\text{\textit{fire}}(Sue), S_0, \text{\textit{do}}(T, S_0))\land \\
\neg\text{\textit{occurs}} \rightarrow \text{\textit{between}}(\text{\textit{quit}}(Sue), S_0, \text{\textit{do}}(T, S_0))
\end{align*}
\]

Execution of query

Use induction and/or simple list processing
State Constraints and the Ramification and Qualification Problems
A State Constraint

\[(\forall s, st). S_0 \leq s \land \text{enrolled}(st, C200, s) \rightarrow \text{enrolled}(st, C100, s)\]

Solution 1: extend successor-state axioms
Enforce next action to be register in missing course

Solution 2: extend transaction-precondition axioms
Ensure that register in C200 is only possible if enrolled in C100
Solution 1: extend successor-state axioms

---

**Original successor-state**

\[
\text{Poss}(a, s) \rightarrow \{ \text{enrolled}(st, c, \text{do}(a, s)) \iff \\
\quad a = \text{register}(st, c) \land \text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c) \}
\]

---

**Extended successor-state**

\[
\text{Poss}(a, s) \rightarrow \{ \text{enrolled}(st, c, \text{do}(a, s)) \iff \\
\quad a = \text{register}(st, c) \\
\quad \lor c = C100 \land a = \text{register}(st, C200) \\
\quad \lor \text{enrolled}(st, c, s) \land a \neq \text{drop}(st, c) \land [c = C200 \rightarrow a \neq \text{drop}(st, C100)] \}
\]
Solution 2: Extend transaction-precondition axioms

Original transaction-precondition

\[
\text{Poss}(\text{register}(st, c), s) \leftrightarrow \\
\{(\forall p).\text{prerequ}(p, c) \rightarrow (\exists g).\text{grade}(st, p, g, s) \land g \geq 50\}
\]

Extended transaction-precondition

\[
\text{Poss}(\text{register}(st, c), s) \leftrightarrow \\
\{(\forall p)[\text{prerequ}(p, c) \rightarrow (\exists g).\text{grade}(st, p, g, s) \land g \geq 50] \\
\land [c = C200 \rightarrow \text{enrolled}(st, C100, s)]\}
\]
\((\forall s, st). S_0 \leq s \land \text{enrolled}(st, C200, s) \rightarrow \text{enrolled}(st, C100, s)\)

can be proofed (e.g., using Induction) to be fulfilled by the extended axioms.
Extensions

- Transaction Logs and Historical Queries
- Complexity of Query Evaluation
- Actualizing Transactions
- Updates in the Logic Programming Context
- Views
- State Constraints and the Ramification and Qualification Problems
Conclusion

Database updates specified using situation calculus

1. Situation Calculus
2. Database Transactions
3. Transaction Logs and Evaluation
4. Proving Properties of Database States
5. Extensions
6. Conclusion

Questions?
References


