On Specifying Database Updates

Survey Talk on the JLP article by Ray Reiter [Rei95]

Jens Bürger[1], Thomas Ruhroth[1] and Emanuel Sallinger[2]

[1]TU Dortmund University, Dortmund, Germany

[2]Vienna University of Technology, Vienna, Austria

Research School FCCOD 2014, Bolzano, Italy

Overview

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

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

The vocabulary also includes the special symbols:

- the predicate **Poss**(*action*, *situation*)
  indicates that an action is possible in a certain situation
- the function **do**(*action*, *situation*)
  describes the resulting situation

Situation Calculus

A logical language over a vocabulary of

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

The vocabulary also includes the special symbols:

- the predicate **Poss**(*action*, *situation*)
  indicates that an action is possible in a certain situation
- the function **do**(*action*, *situation*)
  describes the resulting situation

Situation Calculus

A logical language over a vocabulary of

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

The vocabulary also includes the special symbols:

- the predicate **Poss**(*action*, *situation*)
  indicates that an action is possible in a certain situation
- the function **do**(*action*, *situation*)
  describes the resulting situation

Situation Calculus

A logical language over a vocabulary of

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

The vocabulary also includes the special symbols:

- the predicate **Poss**(*action*, *situation*)
  indicates that an action is possible in a certain situation
- the function **do**(*action*, *situation*)
  describes the resulting situation

Situation Calculus

A logical language over a vocabulary of

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

The vocabulary also includes the special symbols:

- the predicate **Poss**$(action, situation)$
  indicates that an action is possible in a certain situation
- the function **do**$(action, situation)$
  describes the resulting situation

Precondition axioms:

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

Effect axioms:

- $\mathbf{Poss}(repair(r, x), s) \rightarrow \neg broken(x, \mathbf{do}(repair(r, x), s))$
- $\mathbf{Poss}(drop(r, x), s) \wedge$
  $fragile(x) \rightarrow broken(x, \mathbf{do}(drop(r, x), s))$

Precondition axioms:

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

Effect axioms:

- $\mathbf{Poss}(\mathit{repair}(r, x), s) \rightarrow \neg \mathit{broken}(x, \mathbf{do}(\mathit{repair}(r, x), s))$
- $\mathbf{Poss}(\mathit{drop}(r, x), s) \land \mathit{fragile}(x) \rightarrow \mathit{broken}(x, \mathbf{do}(\mathit{drop}(r, x), s))$

Precondition axioms:

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

Effect axioms:

- $\mathbf{Poss}(\mathit{repair}(r, x), s) \rightarrow \neg \mathit{broken}(x, \mathbf{do}(\mathit{repair}(r, x), s))$
- $\mathbf{Poss}(\mathit{drop}(r, x), s) \wedge \mathit{fragile}(x) \rightarrow \mathit{broken}(x, \mathbf{do}(\mathit{drop}(r, x), s))$

Precondition axioms:

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

Effect axioms:

- $\mathbf{Poss}(repair(r, x), s) \rightarrow \neg broken(x, \mathbf{do}(repair(r, x), s))$
- $\mathbf{Poss}(drop(r, x), s) \wedge fragile(x) \rightarrow broken(x, \mathbf{do}(drop(r, x), s))$

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:

- $\mathbf{Poss}(drop(r,x),s) \wedge$
  $color(y,c,s) \rightarrow color(y,c,\mathbf{do}(drop(r,x),s))$
- $\mathbf{Poss}(drop(r,x),s) \wedge \neg broken(y,s) \wedge$
  $[y \neq x \vee \neg fragile(y)] \rightarrow \neg broken(y,\mathbf{do}(drop(r,x),s))$

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:

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

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:

- $\mathbf{Poss}(drop(r,x),s) \wedge$
  $color(y,c,s) \rightarrow color(y,c,\mathbf{do}(drop(r,x),s))$
- $\mathbf{Poss}(drop(r,x),s) \wedge \neg broken(y,s) \wedge$
  $[y \neq x \vee \neg fragile(y)] \rightarrow \neg broken(y,\mathbf{do}(drop(r,x),s))$

Modeling Databases

Some database relations are modeled as fluents:

- $enrolled(student, course, s)$
- $grade(student, course, grade, s)$

Some as atemporals:

- $prereq(prerequisite, course)$

Modeling Databases

Some database relations are modeled as fluents:

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

Some as atemporals:

- $\mathit{prereq}(\mathit{prerequisite}, \mathit{course})$

Modeling Transactions

Transactions (changes to the database) are modeled as actions:

- $\mathit{register}(\mathit{student}, \mathit{course})$
- $\mathit{change}(\mathit{student}, \mathit{course}, \mathit{grade})$
- $\mathit{drop}(\mathit{student}, \mathit{course})$

Modeling Preconditions

Most transactions have particular preconditions:

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

Observe the common syntactic form of these preconditions!

Modeling Preconditions

Most transactions have particular preconditions:

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

Observe the common syntactic form of these preconditions!

Modeling Preconditions

Most transactions have particular preconditions:

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

Observe the common syntactic form of these preconditions!

Modeling Effects

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

- $\mathbf{Poss}(a, s) \rightarrow [enrolled(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = register(st, c) \vee$
  $(enrolled(st, c, s) \wedge a \neq drop(st, c))]$
- $\mathbf{Poss}(a, s) \rightarrow [grade(st, c, g, \mathbf{do}(a, s)) \leftrightarrow$
  $a = change(st, c, g) \vee$
  $(grade(st, c, g, s) \wedge [\forall g'\, g' \neq g \rightarrow a \neq change(st, c, g')])]$

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:

- $\textbf{Poss}(a, s) \rightarrow [\mathit{enrolled}(st, c, \textbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{register}(st, c) \vee$
  $(\mathit{enrolled}(st, c, s) \wedge a \neq \mathit{drop}(st, c))]$
- $\textbf{Poss}(a, s) \rightarrow [\mathit{grade}(st, c, g, \textbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{change}(st, c, g) \vee$
  $(\mathit{grade}(st, c, g, s) \wedge [\forall g'\, g' \neq g \rightarrow a \neq \mathit{change}(st, c, g')])]$

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

The Frame Problem Revisited

- $\mathbf{Poss}(a, s) \rightarrow [\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{register}(st, c) \vee$
  $(\mathit{enrolled}(st, c, s) \wedge a \neq \mathit{drop}(st, c))]$

implies

- $\mathbf{Poss}(a, s) \wedge$
  $a \neq \mathit{register}(st, c) \wedge a \neq \mathit{drop}(st, c)) \rightarrow$
  $[\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow \mathit{enrolled}(st, c, s)]$

"The database relation *enrolled* can *only* be affected by transactions *register* or *drop*."

The Frame Problem Revisited

- $\mathbf{Poss}(a, s) \rightarrow [\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{register}(st, c) \vee$
  $(\mathit{enrolled}(st, c, s) \wedge a \neq \mathit{drop}(st, c))]$

implies

- $\mathbf{Poss}(a, s) \wedge$
  $a \neq \mathit{register}(st, c) \wedge a \neq \mathit{drop}(st, c)) \rightarrow$
  $[\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow \mathit{enrolled}(st, c, s)]$

"The database relation *enrolled* can *only* be affected by transactions *register* or *drop*."

The Frame Problem Revisited

- $\mathbf{Poss}(a, s) \rightarrow [\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{register}(st, c) \vee$
  $(\mathit{enrolled}(st, c, s) \wedge a \neq \mathit{drop}(st, c))]$

implies

- $\mathbf{Poss}(a, s) \wedge$
  $a \neq \mathit{register}(st, c) \wedge a \neq \mathit{drop}(st, c)) \rightarrow$
  $[\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow \mathit{enrolled}(st, c, s)]$

"The database relation *enrolled* can *only* be affected by transactions *register* or *drop*."

The Frame Problem Revisited

- $\mathbf{Poss}(a, s) \rightarrow [\mathit{enrolled}(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = \mathit{register}(st, c) \vee$
  $(\mathit{enrolled}(st, c, s) \wedge a \neq \mathit{drop}(st, c))]$

Succinct representation of the frame axioms is possible because:

- quantification over all transactions
- the assumption that “few” transactions affect a particular database relation

Modeling Queries

What if we want to know

"Is John enrolled in any course after transaction sequence

$drop(John, \mathrm{C100}), register(\mathrm{Mary}, \mathrm{C100})$

from initial state S_0?"

We need to evaluate over our database the formula

$$\exists c\ enrolled(\mathrm{John}, c, \mathbf{do}(register(\mathrm{Mary}, C100), \mathbf{do}(drop(John, \mathrm{C100}), \mathrm{S}_0)))$$

This is called the temporal projection problem.

Modeling Queries

What if we want to know

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

We need to evaluate over our database the formula

$$\exists c\ enrolled(\mathrm{John}, c, \mathbf{do}(register(\mathrm{Mary}, C100), \mathbf{do}(drop(John, \mathrm{C100}), \mathrm{S}_0)))$$

This is called the temporal projection problem.

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

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

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

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

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

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

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

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:

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

A simple formula is a first-order formula that

- does not contain **Poss** or **do**
- does not quantify over states

Simple Formulas

Recall the example:

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

A simple formula is a first-order formula that

- does not contain **Poss** or **do**
- does not quantify over states

Transaction Precondition Axioms

Recall the example:

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

A transaction precondition axiom has the form

$$\forall \vec{x}\, \forall s\, \textbf{Poss}(\textit{transaction}(x_1, \ldots, x_n), s) \leftrightarrow \Pi_{transaction}$$

Transaction Precondition Axioms

Recall the example:

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

A transaction precondition axiom has the form

$$\forall \vec{x}\, \forall s\, \textbf{Poss}(transaction(x_1, \ldots, x_n), s) \leftrightarrow \Pi_{transaction}$$

Successor State Axioms

Recall the example:

- $\mathbf{Poss}(a, s) \rightarrow [enrolled(st, c, \mathbf{do}(a, s)) \leftrightarrow$
  $a = register(st, c) \vee$
  $(enrolled(st, c, s) \wedge a \neq drop(st, c))]$
- $\mathbf{Poss}(a, s) \rightarrow [grade(st, c, g, \mathbf{do}(a, s)) \leftrightarrow$
  $a = change(st, c, g) \vee$
  $(grade(st, c, g, s) \wedge [\forall g'\ g' \neq g \rightarrow a \neq change(st, c, g')])]$

A successor state axiom has the form

$$\forall a \forall s\ \mathbf{Poss}(a, s) \rightarrow \forall \vec{x}\ fluent(x_1, \ldots, x_n, \mathbf{do}(a, s)) \leftrightarrow \Phi_{fluent}$$

Successor State Axioms

Recall the example:

- $\textbf{Poss}(a, s) \rightarrow [\textit{enrolled}(st, c, \textbf{do}(a, s)) \leftrightarrow$
  $a = \textit{register}(st, c) \vee$
  $(\textit{enrolled}(st, c, s) \wedge a \neq \textit{drop}(st, c))]$
- $\textbf{Poss}(a, s) \rightarrow [\textit{grade}(st, c, g, \textbf{do}(a, s)) \leftrightarrow$
  $a = \textit{change}(st, c, g) \vee$
  $(\textit{grade}(st, c, g, s) \wedge [\forall g'\, g' \neq g \rightarrow a \neq \textit{change}(st, c, g')])]$

A successor state axiom has the form

$$\forall a \forall s\, \textbf{Poss}(a, s) \rightarrow \forall \vec{x}\, \textit{fluent}(x_1, \ldots, x_n, \textbf{do}(a, s)) \leftrightarrow \Phi_{\textit{fluent}}$$

The Frame Problem Solved

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

- $\mathbf{Poss}(a, s) \rightarrow [grade(st, c, g, \mathbf{do}(a, s)) \leftrightarrow$
  $a = change(st, c, g) \vee$
  $(grade(st, c, g, s) \wedge [\forall g'\ g' \neq g \rightarrow a \neq change(st, c, g')])]$

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

The Frame Problem Solved

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

- $\mathbf{Poss}(a, s) \rightarrow [grade(st, c, g, \mathbf{do}(a, s)) \leftrightarrow$
  $a = change(st, c, g) \vee$
  $(grade(st, c, g, s) \wedge [\forall g'\, g' \neq g \rightarrow a \neq change(st, c, g')])]$

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

Transaction Logs and Evaluation

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?

Transaction Logs and Evaluation

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

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

$$(\forall s) \neg s < S_0 \tag{1}$$

$$(\forall a, s, s').s < \mathbf{do}(a, s') \leftrightarrow \mathbf{Poss}(a, s') \wedge s \leq s' \tag{2}$$

Legal Transaction Sequences

- Illegal transaction sequences fairly exist:

Example

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

$$(\forall s)\neg s < S_0 \qquad (1)$$

$$(\forall a, s, s').s < \mathbf{do}(a, s') \leftrightarrow \mathbf{Poss}(a, s') \wedge s \leq s' \qquad (2)$$

Legal Transaction Sequences

- Illegal transaction sequences fairly exist:

Example

- *drop*(Sue, $C100$), *change*($Bill$, $C100$, 60)
- Is false, if e.g. **Poss**(*drop*(Sue, $C100$), 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

$$(\forall s)\neg s < S_0 \tag{1}$$

$$(\forall a, s, s').s < \mathbf{do}(a, s') \leftrightarrow \mathbf{Poss}(a, s') \wedge s \leq s' \tag{2}$$

- Common induction principle to be used later on:

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \rightarrow P(\mathbf{do}(a, s))] \rightarrow (\forall s)P(s). \qquad (3)$$

- Compare with the induction axiom for natural numbers:

$$(\forall P).P(0) \wedge (\forall x)[P(x) \rightarrow P(succ(x))] \rightarrow (\forall x)P(x).$$

Legal Transaction Sequences

Induction Principle

- Common induction principle to be used later on:

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

- Compare with the induction axiom for natural numbers:

$$(\forall P).P(0) \land (\forall x)[P(x) \rightarrow P(succ(x))] \rightarrow (\forall x)P(x).$$

Legal Transaction Sequences

Definition of database

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

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

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

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

- Σ: 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

Legal Transaction Sequences

Definition of database

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

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

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

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

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

[1] terms not mentioning any variable

Regression operator $\mathcal{R}$

- *unfolding* operation
- reduce complexity of ground terms[1]
- 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,...:
    $\mathcal{R}^n[G] = \mathcal{R}[\mathcal{R}^{n-1}[G]]$ aso.

[1] terms not mentioning any variable

Theorem [Rei95]:

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

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

precond(τ,s) specifies circumstances under which ground transaction τ is possible in state *s*.

Legal Transaction Sequences

Example: Legality Testing

Consider following transaction sequence:

Example

$\mathit{register}(\mathit{Bill}, C100), \mathit{drop}(\mathit{Bill}, C100), \mathit{drop}(\mathit{Bill}, C100)$

$$\mathcal{R}^0[\mathit{precond}(\mathit{register}(\mathit{Bill}, C100), S_0)] \wedge$$
$$\mathcal{R}^1[\mathit{precond}(\mathit{drop}(\mathit{Bill}, C100), \mathbf{do}(\mathit{register}(\mathit{Bill}, C100), S_0))] \wedge$$
$$\mathcal{R}^2[\mathit{precond}(\mathit{drop}(\mathit{Bill}, C100),$$
$$\mathbf{do}(\mathit{drop}(\mathit{Bill}, C100), \mathbf{do}(\mathit{register}(\mathit{Bill}, C100), S_0)))]$$

Consider following transaction sequence:

Example

$register(Bill, C100), drop(Bill, C100), drop(Bill, C100)$

$$\mathcal{R}^0[precond(register(Bill, C100), S_0)] \wedge$$
$$\mathcal{R}^1[precond(drop(Bill, C100), \mathbf{do}(register(Bill, C100), S_0))] \wedge$$
$$\mathcal{R}^2[precond(drop(Bill, C100),$$
$$\mathbf{do}(drop(Bill, C100), \mathbf{do}(register(Bill, C100), S_0)))]$$

Legal Transaction Sequences

Example: Legality Testing (cont'd)

which is

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

which leads to

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

Legal Transaction Sequences

Example: Legality Testing (cont'd)

which is

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

which leads to

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

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:

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

Reiter's result

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

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

iff

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

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:

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

Reiter's result

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

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

iff

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

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:

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

Reiter's result

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

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

iff

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

Query Evaluation

Example

- Given:
  $\mathbf{T} = \mathit{change}(\mathit{Bill}, C100, 60), \mathit{register}(\mathit{Sue}, C200), \mathit{drop}(\mathit{Bill}, C100)$
- Query:

$$(\exists st).\mathit{enrolled}(st, C200, \mathbf{do}(\mathbf{T}, S_0)) \land$$
$$\neg \mathit{enrolled}(st, C100, \mathbf{do}(\mathbf{T}, S_0)) \land$$
$$(\exists g).\mathit{grade}(st, C200, g, \mathbf{do}(\mathbf{T}, S_0)) \land g \geq 50$$

- $\rightsquigarrow \mathcal{R}^3$ needs to be computed.
- Applying some simplifications (and assume $\mathcal{D}_{S_0} \models C100 \neq C200$):

$$(\exists st).[st = \mathit{Sue} \lor \mathit{enrolled}(st, C200, S_0)] \land$$
$$[st = \mathit{Bill} \lor \neg \mathit{enrolled}(st, C100, S_0)] \land$$
$$[(\exists g).\mathit{grade}(st, C200, g, S_0) \land g \geq 50]$$

Query Evaluation

Example

- Given:
  $\mathbf{T} = \mathit{change}(\mathit{Bill}, C100, 60), \mathit{register}(\mathit{Sue}, C200), \mathit{drop}(\mathit{Bill}, C100)$
- Query:

$$
\begin{aligned}
(\exists st).\mathit{enrolled}(st, C200, \mathbf{do}(\mathbf{T}, S_0))\wedge \\
\neg\mathit{enrolled}(st, C100, \mathbf{do}(\mathbf{T}, S_0))\wedge \\
(\exists g).\mathit{grade}(st, C200, g, \mathbf{do}(\mathbf{T}, S_0)) \wedge g \geq 50
\end{aligned}
$$

- $\rightsquigarrow \mathcal{R}^3$ needs to be computed.
- Applying some simplifications (and assume $\mathcal{D}_{S_0} \models C100 \neq C200$):

$$
\begin{aligned}
(\exists st).[st = \mathit{Sue} \vee \mathit{enrolled}(st, C200, S_0)]\wedge \\
[st = \mathit{Bill} \vee \neg\mathit{enrolled}(st, C100, S_0)]\wedge \\
[(\exists g).\mathit{grade}(st, C200, g, S_0) \wedge g \geq 50]
\end{aligned}
$$

- Recall analogy between natural numbers and database updates:
- let S_0 be identified with *0* and **do**(*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*)
- $IP_{S_0 \leq s \wedge s \leq s'}$
  - (a property holds *between* two states s, s')
- $\rightsquigarrow$ Can be used to prove
  - functionial dependencies (when using *grade*, all the other grades remain unchanged)
  - dynamic integrity constraints (dynamically checking if salary of an employee ever decreases)

Proving Properties of Database States

Induction and the Verification of Integrity Constraints

- Recall analogy between natural numbers and database updates:
- let S_0 be identified with *0* and **do**(*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*)
- $IP_{S_0 \leq s \wedge s \leq s'}$
  - (a property holds *between* two states s, s')
- $\rightsquigarrow$ Can be used to prove
  - functionial 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?
$drop(Mary, C100)$

Property Example: Has some action happened in the history?

Has Sue always worked in Department 13?
$amp(Sue, 13, s)$

Action Example: Has some action happened in a part of the history?

Has Mary dropped the course C100 between situation s and s'?
$drop(Mary, C100)$

Formalization using $<$ operator

Specific Point in History

$(\exists s).S_0 \leq s \wedge s \leq s' \wedge someprop(s)$
$(\exists s).S_0 \leq s \wedge s \leq \mathbf{do}(T, S_0) \wedge someprop(s)$

Whole History

$(\forall s).S_0 \leq s \wedge s \leq s' \rightarrow someprop(s)$
$(\forall s).S_0 \leq s \wedge s \leq \mathbf{do}(T, S_0) \rightarrow someprop(s)$

Part of History

$(occurs - between(a, s, s') \triangleq (\exists s'').s < \mathbf{do}(a, s'') < s'$

Examples formalized

Has Mary dropped the course C100?

$(\exists s, s').S_0 \leq s \wedge s \leq \mathbf{do}(T, S_0) \wedge s = \mathbf{do}(drop(Mary, C100), s')$

Has Sue always worked in Department 13?

$(\forall s).S_0 \leq s \wedge s \leq \mathbf{do}(T, S_0) \rightarrow emp(Sue, 13, s)$

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

$(occurs - between(drop(Mary, C100), s, s')$

Performing Queries - Idea

Transform into “Action-Form”

$emp(Sue, 13, S_0) \wedge$
$\neg occurs - between(fire(Sue), S_0, \mathbf{do}(T, S_0)) \wedge$
$\neg occurs - between(quit(Sue), S_0, \mathbf{do}(T, S_0))$

Execution of query

Use induction and/or simple list processing

State Constraints and the Ramification and Qualification Problems

A State Constraint

$$(\forall s, st).S_0 \leq s \land \mathit{enrolled}(st, C200, s) \rightarrow \mathit{enrolled}(st, C100, s)$$

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

$\mathbf{Poss}(a, s) \rightarrow \{enrolled(st, c, \mathbf{do}(a, s)) \leftrightarrow$
$a = register(st, c) \wedge enrolled(st, c, s) \wedge a \neq drop(st, c)\}$

Extended successor-state

$\mathbf{Poss}(a, s) \rightarrow \{enrolled(st, c, \mathbf{do}(a, s)) \leftrightarrow$
$a = register(st, c)$
$\vee c = C100 \wedge a = register(st, C200)$
$\vee enrolled(st, c, s) \wedge a \neq drop(st, c) \wedge [c = C200 \rightarrow a \neq$
$drop(st, C100)]\}$

Solution 2: Extend transaction-precondition axioms

Original transaction-precondition

$\mathbf{Poss}(\mathit{register}(st, c), s) \leftrightarrow$
$\{(\forall p).\mathit{prerequ}(p, c) \rightarrow (\exists g).\mathit{grade}(st, p, g, s) \wedge g \geq 50\}$

Extended transaction-precondition

$\mathbf{Poss}(\mathit{register}(st, c), s) \leftrightarrow$
$\{(\forall p)[\mathit{prerequ}(p, c) \rightarrow (\exists g).\mathit{grade}(st, p, g, s) \wedge g \geq 50]$
$\wedge[c = C200 \rightarrow \mathit{enrolled}(st, C100, s)]\}$

Original Constraint

$(\forall s, st).S_0 \leq s \wedge \mathit{enrolled}(st, C200, s) \rightarrow \mathit{enrolled}(st, C100, s)$

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

- Fangzhen Lin, *Situation calculus*, Handbook of Knowledge Representation (Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, eds.), Elsevier, 2008.
- John McCarthy, *Situations, actions and causal laws*, Semantic Information Processing (1968), 410–417.
- John McCarthy and Patrick Hayes, *Some philosophical problems from the standpoint of artificial intelligence*, Machine Intelligence (1969), 463–502.
- Raymond Reiter, *On specifying database updates*, J. Log. Program. **25** (1995), no. 1, 53–91.
- ———, *Knowledge in action*, MIT Press, 2001.