Abstract

The concept of referential integrity plays an important role in local "repairs" after a user-requested update.

List of Theorems and Friends.

List of Definitions.

List of Examples.

1 Introduction

LongTh α **(bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla)***:gaga.* ♡

Proposition 1 *gaga*

> *Dat isn quote.*

□

Eine Referenz auf Prop 1: Proposition 1 on page 2. The introduction section starts on page 2.

Lemma 1

$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \int_\gamma f(z)\,dz := \int_a^b f(\gamma(t))\gamma'(t)\,dt \tag{1}$$

$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \int_\gamma f(z)\,dz := \int_a^b f(\gamma(t))\gamma'(t)\,dt \tag{2}$$

□

Lemma 2

$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \int_\gamma f(z)\,dz := \int_a^b f(\gamma(t))\gamma'(t)\,dt \tag{3}$$

$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \int_\gamma f(z)\,dz := \int_a^b f(\gamma(t))\gamma'(t)\,dt \tag{4}$$

■

Definition 1

$$f(z) = \frac{1}{2\pi i} \int\limits_{\partial D} \frac{f(\zeta)}{\zeta - z} d\zeta \tag{5}$$

$$= \frac{1}{2\pi} \int\limits_0^{2\pi} f(z_0 + re^{it}) dt \tag{6}$$

□

Proof

$$\begin{aligned} f(z) &= \frac{1}{2\pi i} \int\limits_{\partial D} \frac{f(\zeta)}{\zeta - z} d\zeta \\ &= \frac{1}{2\pi} \int\limits_0^{2\pi} f(z_0 + re^{it}) dt \end{aligned}$$

■

$$\eta_E(\chi, \psi)\, a = \star_E \chi \wedge \psi \tag{7}$$

Proposition 2 *Dat isn quote.*

> *The postscript for accepted papers that have not yet appeared in print is also available. This ensures very rapid availability. As an example, the paper "Transactional Client-Server Cache Consistency: Alternatives and Performance" by Michael Franklin, Michael Carey and Miron Livny was submitted on October 1995, went through two rounds of review and revision, was accepted, and appeared in the ACM Digital Library in January 1997, only fifteen months later.* □

Prop *bla* ♡

Das Whatsit-Problem, erster Teil: In der ersten Umgebung wird kein Endzeichen gesetzt, in der zweiten funktioniert es:

Prop

1. *...*
2. *...x* ■

Prop

1. *...*
2. *...* ■

Härtetest fuer Filebehandlung:
Theoremname per Token, Endzeichen: Mehrmals Umdefiniertes qed-zeichen, ausserdem ein Eintrag ins Theoremfile, der ein späteres Label referenziert:

```
\addtotheoremfile[Proposition]{Das ist die letzte Prop.
vor Prop. \ref{PropDieNaechste}}
```

Prop

1. *...*
2. *...*

Proposition 3 (x)

1. ...

2. ...x

Proposition 4 (y)

1. ...

2. ...y

κ-Theorem 1 (1st κ-Theorem) *That's the first Kappa-Theorem.* $\begin{matrix} a \\ b \end{matrix}$

The first Kappa-Theorem has been given in κ-Theorem 1.

Systematically test with the center environment:

centered

Plain 1 (title)

1 Center // Center

$\frac{a}{b}$

Break 1 (title)

2 Center // Center

$\frac{a}{b}$

1 MarginBreak (title)

3 Center // Center

$\frac{a}{b}$

1 ChangeBreak (title)

4 Center // Center

$\frac{a}{b}$

Plain 2 (title)

- *5 itemize*
- *itemize*

$\frac{a}{b}$

Break 2 (title)

- *6 itemize*
- *itemize*

$\frac{a}{b}$

2 MarginBreak (title)

- *7 itemize*
- *itemize*

$\frac{a}{b}$

Break 3 (title)

quote

$\frac{a}{b}$

3 MarginBreak (title)

quote

$\frac{a}{b}$

Break 4 (title)

```
verbatim
```

$\frac{a}{b}$

4 MarginBreak (title)

```
verbatim
```

$\begin{matrix} a \\ b \end{matrix}$

centering in items looks a bit weird, but that has nothing to do with ntheorem:

Break 5 (title)

- *center in item*

- *center in item* $\begin{matrix} a \\ b \end{matrix}$

Example 1 Consider the database with referential actions as depicted in Figure **??**. For this example, assume that *all dotted parts are empty*. Let $\triangleright\mathsf{del}{:}R_A(a)$[1] be a user request to delete the tuple (a) from relation R_A. Depending on the order of execution of referential actions, one of two different final states may be reached:

(1) If execution follows the path $R_A \rightsquigarrow R_C \rightsquigarrow R_D$, the tuple $R_C(a, c)$ cannot be deleted: Since $R_D(a, b, c)$ references $R_C(a, c)$, the referential action for R_D restricts the deletions of $R_C(a, c)$. This in turn also blocks the deletion of $R_A(a)$. Consequently, the user request $\triangleright\mathsf{del}{:}R_A(a)$ is rejected, and the database state remains unchanged, ie $D' = D$.

(2) If execution follows the path $R_A \rightsquigarrow R_B \rightsquigarrow R_D$, the tuple $R_B(a, b)$ and – as a consequence – $R_D(a, b, c)$ are requested for deletion. Hence, the trigger for $R_D.(X, Z) \rightarrow R_C.(X, Z)$ "assumes" that $R_D(a, b, c)$ is deleted, thus no referencing tuple exists in R_D. Thus, all deletions can be executed, resulting in the new database state $D' = \emptyset$.

gaga

gaga

□

Example 2 Consider the database as depicted in Figure **??** and assume the user requests $\{\triangleright\mathsf{del}{:}R_A(a), \triangleright\mathsf{del}{:}R_A(b)\}$ are

gaga

given. $\triangleright\mathsf{del}{:}R_A(a)$ is not admissible since $R_E(a)$ blocks $\triangleright\mathsf{del}{:}R_A(a)$. However, the other request, $\triangleright\mathsf{del}{:}R_A(b)$, could be executed without violating any *ric* by deleting $R_A(b)$, $R_B(b, b)$, $R_C(b, c)$ and $R_D(b, b, c)$.

$$a = b \tag{8}$$

$$c = d \tag{9}$$

□

[1] The triangle "$\triangleright$" denotes *external* (ie, user-defined) requests.

Example 3

$$\begin{aligned} a &= b \\ c &= eqnarray* \end{aligned} \qquad \square$$

$$test \tag{10}$$

With theoremstyle break, if the environment is labelled and written as

```
\begin{XLemma}[Whatsit ?]\label{whatsit}
```

XLemma α *(Whatsit ?):*

you see, there is a leading space . . .

If a percent (comment) is put directly after he label, the space disappears.

From the predefined styles, this is exactly the case for the break-styles. That's no bug, it's LaTeX-immanent. ♡

Example 4 The "diamond" in Figure **??** results in a "dispute" between blockings and deletions: Given the user request $\triangleright$del:$R_A(a)$, the delete requests req_del for $R_A(a)$,

$$a = b \tag{11}$$

$$c = d \tag{12}$$

$R_B(a,b)$, $R_C(a,c)$, $R_D(a,b,c)$, as well as the blockings blk_del for $R_A(a)$, $R_C(a,c)$ will be *undefined* in the well-founded model.

Lemma 3 *Das ist ein Lemma innerhalb eines Beispiels.* $\square$

Looking at the database in Figure **??** with the user requests $\{\triangleright\text{del}{:}R_A(a), \triangleright\text{del}{:}R_A(b)\}$, we find that the blockings for $R_A(a)$ and $R_C(a,c)$ are *true* in the well-founded model (due to the referencing tuple $R_E(a)$) and thus $R_A(a)$, $R_C(a,c)$ cannot be deleted. In contrast, for $R_A(b), R_B(b,b), R_C(b,c)$ and $R_D(b,b,c)$ there are undefined delete requests, and for $R_A(b), R_B(b,b)$ and $R_D(b,b,c)$ there are also undefined blockings. $\square$

Definition 2 Player I can move from $R(\bar{x})$ to $\triangleright$del:$R'(\bar{x}')$:$\Leftrightarrow$

"there is a finite sequence of ric's with `ON DELETE CASCADE` *triggers leading from $\triangleright$del:$R'(\bar{x}')$ to $R(\bar{x})$ in D."*

Player II can move from $\triangleright$del:$R(\bar{x})$ to $R'(\bar{x}')$:$\Leftrightarrow$

"$R'(\bar{x}')$ is blocked by some `ON DELETE RESTRICT` *trigger, and there a finite (possibly empty) sequence of ric's with* `ON DELETE CASCADE` *triggers leading from $\triangleright$del:$R(\bar{x})$ to $R(\bar{x})$." xxx*

$$testequation \tag{13}$$ $\square$

Theorem 2 *For every tuple $R(\bar{x})$ there is*

- *$R(\bar{x})$ is won or drawn iff simultaneous execution of all user delete requests $\triangleright$del:$R'(\bar{x}')$ s.t. $R'(\bar{x}')$ is won or drawn does not violate any ric and deletes $R(\bar{x})$.*
- *$R(\bar{x})$ is lost iff it is not possible with the given set of user delete requests to delete $R(\bar{x})$ without violating any ric.* □

ausserhalb Theorem !! muss normalfont sein.

PROOF
- $R(\bar{x})$ is won or drawn if there is a user request $\triangleright$del:$R'(\bar{x}')$ s.t. del:$R'(\bar{x}')$ is also won or drawn and there is a sequence of `on delete cascade`-triggers from $R'(\bar{x}')$ to $R(\bar{x})$ and either there is no tuple which (transitively) restricts this user request or all tuples which (transitively) restrict this user request are either won or drawn. Due to $\triangleright$del:$R'(\bar{x}')$ and the above sequence of cascading triggers, $R(\bar{x})$ is deleted when $\triangleright$del:$R'(\bar{x}')$ is executed. With the same argument, all tuples which potentially restrict this execution are also removed, thus no *ric* of the form `on delete restrict` is violated. Since all tuples reachable from a user request which is won or lost by `on delete cascade`-triggers are also won resp. lost, all of them will also be deleted, thus no *ric* of the form `on delete cascade` is violated.
- A tuple $R(\bar{x})$ is lost in n moves if there is either ($n = 0$) no user request $\triangleright$del:$R'(\bar{x}')$ s.t. there is a sequence of `on delete cascade`-triggers from $R'(\bar{x}')$ to $R(\bar{x})$ or every such user request is lost. The latter means that there is some tuple $R''(\bar{x}'')$ transitively restricting the deletion of $R'(\bar{x}')$ which is lost in less than n moves. By induction hypothesis, there is no way to delete $R''(\bar{x}'')$, thus there is also no way to delete $R(\bar{x})$. ■

2 Next Section

Example 5 Consider again the "diamond" in Figure **??**. The positions are $R_A(a)$, $R_B(a,b)$, $R_C(a,c)$, $R_D(a,b,c)$, and $\triangleright$del:$R_A(a)$.

`I` can move from any position in $\{R_A(a), R_B(a,b), R_C(a,c), R_D(a,b,c)\}$ to $\triangleright$del:$R_A(a)$, while `II` can move from $\triangleright$del:$R_A(a)$ to $R_D(a,b,c)$. Thus, after `I` has started the game moving to $\triangleright$del:$R_A(a)$, `II` will answer with the move to $R_D(a,b,c)$ and so on. Hence the game is drawn for all start positions of `I`.

In contrast, if $R_E(a)$ is added to the database in Figure **??**, there is an additional move from $\triangleright$del:$R_A(a)$ to $R_E(a)$ for `II`, who now has a winning strategy: by moving to $R_E(a)$, there is no possible answer for `I`, so `I` loses. By Theorems 3 and **??**, $R_A(a)$ cannot be deleted. □

Theorem 3 *The well-founded model and the game-theoretic semantics correspond as follows:*

- *I wins at* $R(\bar{x})$ *iff* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) = true$,
- *II wins at* $R(\bar{x})$ *iff* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) = false$, *and*
- $R(\bar{x})$ *is drawn iff* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) = undef$. □

The proof is postponed to Section **??**, where a computational characterization of $\mathcal{W}(P, D, U_\triangleright)$ is presented.

Theorem 4
- *Each internal delete request such that* $v := \mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) \in \{true, undef\}$ *is* founded *by some user delete request* $\triangleright\mathsf{del}{:}R'(\bar{x}')$ *s.t.* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) = v$, *ie there is a chain of references from* $R(\bar{x})$ *to* $R'(\bar{x}')$ *in* D *using* `ON DELETE CASCADE` *triggers.*
- *For all tuples laying on this chain also* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R''(\bar{x}'')) = v$.
- *For every* $\triangleright\mathsf{del}{:}R'(\bar{x}') \in U_\triangleright$,
  - $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R'(\bar{x}')) = true \;\Rightarrow\;$ *for all* $R(\bar{x})$ *which are reachable from* $R'(\bar{x}')$ *by a chain of* `ON DELETE CASCADE` *triggers,* $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R'(\bar{x}')) = true$, *and*
  - $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R'(\bar{x}')) = undef \;\Rightarrow\;$ *for all such* $R(\bar{x})$, $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R'(\bar{x}')) \in \{true, undef\}$. □

Definition 3 In each case, there must be a first move of `I` to some $\triangleright\mathsf{del}{:}R'(\bar{x}')$ which is also won/drawn, thus $\mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R'(\bar{x}')) = v$. Obviously, all positions in between are also won/drawn, and all positions from where `I` can move to a won/drawn user request, are also won/at least drawn. □

Theorem 5

$$[S{+}1]\ \mathsf{req\_del}{:}R(\bar{X}) \leftarrow\ \triangleright\mathsf{del}{:}R(\bar{X}), [S]\ \neg\mathsf{blk\_del}{:}R(\bar{X}). \qquad (I^A)$$

% $R_C.\vec{X} \to R_P.\vec{Y}$ `ON DELETE CASCADE:`

$$[S{+}1]\ \mathsf{req\_del}{:}R_C(\vec{X}, \bar{X}) \leftarrow\ R_C(\vec{X}, \bar{X}), \vec{X} = \vec{Y}, [S{+}1]\ \mathsf{req\_del}{:}R_P(\vec{Y}, \bar{Y}), \qquad (DC_1^A)$$

$$[S{+}1]\ \mathsf{blk\_del}{:}R_P(\vec{Y}, \bar{Y}) \leftarrow\ R_P(\vec{Y}, \bar{Y}), \vec{X} = \vec{Y}, [S{+}1]\ \mathsf{blk\_del}{:}R_C(\vec{X}, \bar{X}). \qquad (DC_1^A)$$

% $R_C.\vec{X} \to R_P.\vec{Y}$ `ON DELETE RESTRICT:`

$$[S{+}1]\ \mathsf{blk\_del}{:}R_P(\vec{Y}, \bar{Y}) \leftarrow\ R_P(\vec{Y}, \bar{Y}), R_C(\vec{X}, \bar{X}), \vec{X}{=}\vec{Y}, [S]\ \neg\mathsf{req\_del}{:}R_C(\vec{X}, \bar{X}). \qquad (DR^A)$$

□

Proof of **Theorem 3**. First, we prove the following

Lemma 4 • *I wins at $R(\bar{x})$ within n rounds iff $\mathcal{M}(P_A, D, U_\triangleright) \models [2n]$ req_del:$R(\bar{x})$.*

• *II wins at $R(\bar{x})$ within n rounds iff $\mathcal{M}(P_A, D, U_\triangleright) \models [2n-1]$ $\neg$req_del:$R(\bar{x})$.*

gagalemma □

PROOF (All subproofs below can be extended to "iff", but for better readability, this is not always formulated exactly.)

II wins in one round starting at $R(\bar{x})$ iff Player I cannot move to a user request, ie if the deletion of $R(\bar{x})$ is unfounded. That is the case iff in the first overestimate of P_A, $R(\bar{x})$ is not requested for deletion: $\mathcal{M}(P_A, D, U_\triangleright) \models [1]$ $\neg$req_del:$R(\bar{x})$.

I wins in one round at $R(\bar{x})$ iff the deletion of $R(\bar{x})$ is founded by some user delete request $\triangleright$del:$R'(\bar{x}')$, and II cannot move from $\triangleright$del:$R'(\bar{x}')$. This is the case, if there is no `ON DELETE CASCADE` chain from $R'(\bar{x}')$ to a tuple $R''(\bar{x}'')$ which is restricted by some other tuple. Thus, in this case, in the first overestimate of P_A, the deletions of $R''(\bar{x}'')$ and $R'(\bar{x}')$ are not blocked: $\mathcal{M}(P_A, D, U_\triangleright) \models [1]$ $\neg$blk_del:$R'(\bar{x}')$. Then, since there is a user delete request $\triangleright$del:$R'(\bar{x}')$, $\mathcal{M}(P_A, D, U_\triangleright) \models [2]$ req_del:$R'(\bar{x}')$ and $\mathcal{M}(P_A, D, U_\triangleright) \models [2]$ req_del:$R(\bar{x})$.

The induction step follows the same line of argumentation:

II wins in $n+1$ rounds at $R(\bar{x})$ iff for all moves to some $\triangleright$del:$R'(\bar{x}')$ of I, he can move to some tuple $R''(\bar{x}'')$ which he wins in n rounds: $\mathcal{M}(P_A, D, U_\triangleright) \models [2n-1]$ $\neg$req_del:$R''(\bar{x}'')$ by induction hypothesis. Thus, since there is a move from $\triangleright$del:$R'(\bar{x}')$ to $R''(\bar{x}'')$, there are triggers `ON DELETE RESTRICT` and `ON DELETE CASCADE` s.t. $\mathcal{M}(P_A, D, U_\triangleright) \models [2n]$ blk_del:$R'(\bar{x}')$. Since this is the case for all $R'(\bar{x}')$ where I can move to from $R(\bar{x})$, $\mathcal{M}(P_A, D, U_\triangleright) \models [2n+1]$ $\neg$req_del:$R''(\bar{x}'')$.

I wins in $n+1$ rounds at $R(\bar{x})$ if there is a $R'(\bar{x}')$ he can move to s.t. for all positions $R''(\bar{x}'')$ where II can move to from $R'(\bar{x}')$, II will lose in at most n rounds. By induction hypothesis, for all those $R''(\bar{x}'')$, $\mathcal{M}(P_A, D, U_\triangleright) \models [2n]$ req_del:$R''(\bar{x}'')$. Thus, $\mathcal{M}(P_A, D, U_\triangleright) \models [2n+1]$ $\neg$blk_del:$R'(\bar{x}')$ and $\mathcal{M}(P_A, D, U_\triangleright) \models [2n+2]$ req_del:$R(\bar{x})$. ■

From the previous lemma, Theorem 3 follows immediately: Since even-numbered states are underestimates, there is an n such that $\mathcal{M}(P_A, D, U_\triangleright) \models [2n]$ req_del:$R(\bar{x})$ iff $\mathcal{W}(P, D, U_\triangleright) \models$ req_del:$R(\bar{x})$, and on the other hand, since odd-numbered states are overestimates, there is an n such that $\mathcal{M}(P_A, D, U_\triangleright) \models [2n+1]$ $\neg$req_del:$R''(\bar{x}'')$ iff $\mathcal{W}(P, D, U_\triangleright) \models \neg$req_del:$R''(\bar{x}'')$. ■

• Underlying Definition

Definition 4 Stable Model [**?**, **?**] Let M_P denote the minimal model of a positive program P. For an interpretation I, let P/I denote the reduction of P wrt. I. Then, an interpretation I is a *stable model* iff $M_{P/I} = I$ (ie a stable model reproduces itself[2]). □

[2] a stable model is a stable model is a stable model.

Fact: The well-founded model agrees on all definite atoms with every stable model.
In general, not every program has a stable model. But, here, the following theorem develops:

- Theorem

Theorem 6 *For every database D, and every set $U_\triangleright$ of user delete requests, there exists at least one stable model.* □

From the well-founded model $\mathcal{W}(P, D, U_\triangleright)$, two distinguished stable models which agree with $\mathcal{W}(P, D, U_\triangleright)$ on all definite atoms (thus, if $\mathcal{W}$ is total, there is exactly one stable model) are induced by the following policies:

Restrictive: Count all delete requests $\mathsf{req\_del}{:}R(\bar{x})$ s.t. $\triangleright\mathsf{del}{:}R(\bar{x}) \in U_\triangleright$ and $\mathcal{W}(\mathsf{req\_del}{:}R(\bar{x})) = \mathit{undef}$ with the false ones.

Permissive: Count all delete requests $\mathsf{req\_del}{:}R(\bar{x})$ s.t. $\triangleright\mathsf{del}{:}R(\bar{x}) \in U_\triangleright$ and $\mathcal{W}(\mathsf{req\_del}{:}R(\bar{x})) = \mathit{undef}$ with the true ones.

Theorem 7

- *The unique stable model satisfying the restrictive policy is $\mathcal{S}_R$:*
  $\mathcal{S}_R(P, D, U_\triangleright) \models \mathit{req\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathit{req\_del}{:}R(\bar{x})) = \mathit{true}$,
  $\mathcal{S}_R(P, D, U_\triangleright) \models \mathit{blk\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathit{blk\_del}{:}R(\bar{x})) \in \{\mathit{true}, \mathit{undef}\}$,
- *The unique stable model satisfying the permissive policy is $\mathcal{S}_P$:*
  $\mathcal{S}_P(P, D, U_\triangleright) \models \mathit{req\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathit{req\_del}{:}R(\bar{x})) \in \{\mathit{true}, \mathit{undef}\}$,
  $\mathcal{S}_P(P, D, U_\triangleright) \models \mathit{blk\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathit{blk\_del}{:}R(\bar{x})) = \mathit{true}$. □

Definition 5 A set Δ of internal delete requests is called

1. *founded wrt. $U_\triangleright$, D and RA* if every $\mathsf{req\_del}{:}R(\bar{x}) \in \Delta$ is *founded* by some $\triangleright\mathsf{del}{:}R'(\bar{x}') \in U_\triangleright$, ie there is a chain of references from $R(\bar{x})$ to $R'(\bar{x}')$ in D using `ON DELETE CASCADE` triggers from RA,
2. *safe wrt. D and RI* if all *ric*s in RI are satisfied in the new database $D' := D \pm \Delta$.

For a set $U_\triangleright$ of user requests, the set of *induced internal updates*, $\Delta(U_\triangleright)$, is the minimal set M satisfying the following conditions:

1. for all $\mathsf{req\_del}{:}R(\bar{x})$ s.t. $\triangleright\mathsf{del}{:}R(\bar{x}) \in U_\triangleright$, $\mathsf{req\_del}{:}R(\bar{x}) \in M$,
2. if $\mathsf{req\_del}{:}R(\vec{x}, \bar{x}) \in M$, $Q(\vec{y}, \bar{y}) \in D$, $Q.\vec{x} \to R.\vec{y}$ `ON DELETE CASCADE` $\in RA$, and $\vec{x} = \vec{y}$ then $\mathsf{req\_del}{:}Q(\vec{y}, \bar{y}) \in M$.

A set U of user requests is *acceptable* if $\Delta(U)$ is safe. qed q.e.d

Lemma 5 *For a set U of external requests, $\Delta(U)$ is founded. If $\Delta(U)$ is safe, then $D' := D \pm \Delta$ is the database obtained by applying the set U of user requests on the database D and D' satisfies.* □

Lemma 6 *verbatim:*

```
What about tomorrows children \\
what about the world they live in \\
what about the way we live today \\
tell me \\
What about tomorrows children \\
we can do a lot to help them \\
and love tomorrows child today
```

□

Lemma 7 *center:*

What about tomorrows children
what about the world they live in
what about the way we live today
tell me
What about tomorrows children
we can do a lot to help them
and love tomorrows child today

□

XLemma *β (title in text):*

It also works in the
center
environment. ♡

Satz 1 *Eine Menge von Aktionen ist homomorph genau dann wenn f"ur ihre Spezifikation durch partielle Interpretationen f"ur alle* $\mathrm{a}, \mathrm{b} \in$ Aktionen *folgendes gilt:*

$$Spec(\mathrm{a} \vee \mathrm{b}) = \mathrm{Spec(a)} \cup \mathrm{Spec(b)} \cup \mathrm{Spec(a||b)}$$
$$Spec(\mathrm{a}^*) = \{(\mathcal{P}, \mathcal{Q}) \mid \exists n : (\mathcal{P}, \mathcal{Q}) \in Spec(\underbrace{\mathrm{a} \times \ldots \times \mathrm{a}}_{n-\mathrm{mal}}) \text{ und}$$
$$\text{es gibt kein } (\mathcal{I}, \mathcal{J}) \in Spec(\mathrm{a}) \text{ so da"s } \mathcal{Q} \cup \mathcal{I} \text{ konsistent ist.}$$

□

In the following, let $\mathcal{S}$ be $\mathcal{S}_R$, $\mathcal{S}_P$, or one of the stable models obtained by the above remark.

Theorem 8 (Correctness) *Let* $\Delta_\mathcal{S} := \{\mathit{req\_del}{:}R(\bar{x}) \mid \mathcal{S} \models \mathit{req\_del}{:}R(\bar{x})\}$ *and* $U_\mathcal{S} := \{\mathit{req\_del}{:}R(\bar{x}) \mid \mathcal{S} \models \mathit{req\_del}{:}R(\bar{x}) \wedge \triangleright\mathit{del}{:}R(\bar{x})\} \subseteq U_\triangleright$. *Then* $\Delta_\mathcal{S}$ *is founded wrt.* $U_\mathcal{S}$, D, *and* RA *and safe wrt.* D *and* RI, *and* $\Delta_\mathcal{S} = \Delta(U_\mathcal{S})$. □

PROOF Foundedness follows directly from Theorem 4.
$\Delta_\mathcal{S} \subseteq \Delta(U_\mathcal{S})$ follows from foundedness, and $\Delta_\mathcal{S} \supseteq \Delta(U_\mathcal{S})$ follows from safeness: otherwise some *ric* encoded as `on delete cascade` would be violated. ■

Theorem 9 (Maximality) *For a given set* $U_\triangleright$ *of user requests, the permissive variant determines the maximal acceptable subset.* □

PROOF Und, just for fun, auch dieser Proof wird mit einem qed anstelle der black Box abgeschlossen: Sie ist auch rückwirkend in das displaymath:

$$\mathcal{S}_P \models \neg\mathsf{req\_del}{:}R(\bar{x}) \overset{\text{policy}}{\Leftrightarrow} \mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) = \mathit{false} \overset{\text{Th. } 3}{\Leftrightarrow}$$
$$\mathcal{R}(\bar{x}) \text{ is lost } \overset{\text{Th. } 2}{\Leftrightarrow} \triangleright\mathsf{del}{:}R(\bar{x}) \text{ is inadmissible.}$$

■

By defining the above policies, the *local* ambiguities of the operational approach are exploited and turned into controlled, global alternatives.

Definition 6 The labeled dependency graph $\mathcal{G}(P)$ of a Statelog program P is defined as follows. Its vertices are the relation names occurring in P. For every rule

$$[S_0]\ H(\bar{X}_0) \leftarrow\ [S_1]\ B_1(\bar{X}_1), \ldots, [S_n]\ B_1(\bar{X}_n)\ .$$

of P, $\mathcal{G}(P)$ contains for every $i = 1, \ldots, n$

- a negative edge $A_i \overset{l_i,\neg}{\rightarrow} H$, if B_i is a negative literal $\neg A_i(\bar{X})$
- a positive edge $B_i \overset{l_i}{\rightarrow} H$ otherwise.

Here, the label $l_i := S_0 - S_i \geq 0$ is the "gap" between states; it may be omitted for $l = 0$.

A cycle of $\mathcal{G}(P)$ involving only edges with $l = 0$ is called a *local cycle*. A program P is called *state-stratified* if no local cycle of $\mathcal{G}(P)$ contains a negative edge. □

DasNeueTheorem (Termination)
For every database D *and every set* $U_\triangleright$ *of user delete requests, there is a unique final state* $n_{final} \leq |U_\triangleright| + 1$, *ie for all* $k < n_{final}$: $\mathcal{M}(P_S, D, U_\triangleright) \models [k]\,\mathit{running}$, *and for all* $k \geq n_{final}$: $\mathcal{M}(P_S, D, U_\triangleright) \models [k]\,\neg\mathit{running}$. $\begin{smallmatrix} a \\ b \end{smallmatrix}$

Lemma 8 *The model* $\mathcal{M}(P_A, D, U_\triangleright)$ *corresponds to* $\mathcal{M}(P_S, D, U_\triangleright)$ *as follows:*

1. $\mathcal{M}(P_A, D, U_\triangleright) \models [2n]\ \mathit{blk\_del}{:}R(\bar{x}) \ \Leftrightarrow\ \mathcal{M}(P_S, D, U_\triangleright) \models [n]\ \mathit{blk\_del}{:}R(\bar{x})$.
2. $\mathcal{M}(P_A, D, U_\triangleright) \models [2n{+}1]\ \mathit{req\_del}{:}R(\bar{x}) \ \Leftrightarrow\ \mathcal{M}(P_S, D, U_\triangleright) \models [n]\ \mathit{req\_del}{:}R(\bar{x})$. □

PROOF P_S and P_A differ in the rules (I^S) and (I^A): While (I^A) derives internal delete requests in [S+1] from unblocked user requests in [S], (I^S) already establishes these in the *current* state [S].

In [0] neither program derives blockings $\mathsf{blk\_del}{:}R(\bar{x})$; hence we have an underestimate of the final set of blockings. From this, both programs derive an overestimate of delete requests $\mathsf{req\_del}{:}R(\bar{x})$. Due to rules (I^S) and (I^A) these overestimates are computed in [S] and [S+1] by (I^S) and (I^A), respectively. Using these overestimates, the next sets of underestimates $\mathsf{blk\_del}{:}R(\bar{x})$ are derived in [1] for P_S, and in [2] for P_A. Applied inductively, this argument concludes the proof. ∎

DasNeueTheorem

1. $\mathcal{M}(P_S, D, U_\triangleright) \models [n_{final}]\, \mathsf{req\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathsf{req\_del}{:}R(\bar{x})) \in \{true, undef\}$.
2. $\mathcal{M}(P_S, D, U_\triangleright) \models [n_{final}]\, \neg\mathsf{blk\_del}{:}R(\bar{x}) \;\Leftrightarrow\; \mathcal{W}(P, D, U_\triangleright)(\mathsf{blk\_del}{:}R(\bar{x})) \in \{false, undef\}$.^a_b

List of Definitions.

List of Examples.