Understanding the Global Semantics of Referential Actions using Logic Rules

WOLFGANG MAY
Institut für Informatik, Universität Freiburg, Germany
and
BERTRAM LUDÄSCHER
San Diego Supercomputer Center, University of California San Diego, USA

The electronic appendix contains an example for the logic-programming characterization of referential actions in case of update operations hat has been given in Section 4.2. Additionally, it describes the *detailed* game-theoretic characterization of that case that has been sketched in Section 4.3, illustrates it by an example, and shows its equivalence with the logic programming characterization.

A. EXAMPLE FOR THE LOGIC PROGRAMMING CHARACTERIZATION OF REFERENTIAL ACTIONS WITH UPDATES

In Section 4.2, a logic programming characterization for full referential actions, including modifications (i.e. `ON UPDATE CASCADE`) has been given. The following example instantiates the logic rules for a given situation.

Example 12 *Modifications: Diamond.* Consider again Figure 1, all references labeled with `ON UPDATE OF PARENT CASCADE` and `ON UPDATE OF CHILD NO ACTION`, which is a completely plausible setting. Consider the external modification request $\triangleright\mathsf{mod}\_R_1(1/n, (a, \ldots))$. Among many others, we have the following rules:
External modification on R_1 (EXT_1):

$$\mathsf{mod}\_R_1(M, \bar{X}) \leftarrow \ \triangleright\mathsf{mod}\_R_1(M, \bar{X}), \neg\ \mathsf{blk\_mod}\_R_1(M, \bar{X}).$$

From $R_2.1 \rightarrow R_1.1$ `ON UPDATE OF PARENT CASCADE` ($Refd$), (MPC_1), (MPC_2): Propagate the modification of the primary key $R_1.1$ downwards to the foreign key $R_2.1$ of the child tuple (if possible, otherwise block the update), and do the

bookkeeping that the new value of $R_1.1$ is referenced:

$$
\begin{array}{ll}
\mathsf{new\_refd}\_R_1.1\_\mathsf{by}\_R_2.1(\bar{V}) & \leftarrow R_2(\bar{X}),\ \ \mathsf{chg}\_R_2.1(M,\bar{X}),\ M(\bar{X})[1]=\bar{V}. \\
\mathsf{prp}\_R_1.1 \rightsquigarrow R_2.1(M_C,\bar{X}) & \leftarrow \mathsf{chg}\_R_1.1(M_P,\bar{Y}),\ R_2(\bar{X}),\ \bar{X}[1]=\bar{Y}[1], \\
 & \quad M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_2.1(M_C,\bar{X}) & \leftarrow \mathsf{pot\_chg}\_R_1.1(M_P,\bar{Y}),\ R_2(\bar{X}),\ \bar{X}[1]=\bar{Y}[1], \\
 & \quad M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{blk\_chg}\_R_1.1(M_P,\bar{Y}) & \leftarrow \mathsf{pot\_chg}\_R_1.1(M_P,\bar{Y}), \\
 & \quad \mathsf{blk\_prop}\_R_1.1 \rightsquigarrow R_2.1\,(M_C,\bar{X}), \\
 & \quad \bar{X}[1]=\bar{Y}[1], \\
 & \quad M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{blk\_prop}\_R_1.1 \rightsquigarrow R_2.1(M,\bar{X}) & \leftarrow \mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_2.1(M,\bar{X}), \\
 & \quad \mathsf{blk\_chg}\_R_2.1\ (M,\bar{X}).
\end{array}
$$

Analogously, from $R_3.1 \rightarrow R_1.1$ `ON UPDATE OF PARENT CASCADE` (*Refd*), (MPC_1), (MPC_2):

$$
\begin{array}{ll}
\mathsf{new\_refd}\_R_1.1\_\mathsf{by}\_R_3.1(\bar{V}) & \leftarrow R_3(\bar{X}),\ \ \mathsf{chg}\_R_3.1(M,\bar{X}),\ M(\bar{X})[1]=\bar{V}. \\
\mathsf{prp}\_R_1.1 \rightsquigarrow R_3.1(M_C,\bar{X}) & \leftarrow \mathsf{chg}\_R_1.1(M_P,\bar{Y}),\ R_3(\bar{X}),\ \bar{X}[1]=\bar{Y}[1], \\
 & \quad M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_3.1(M_C,\bar{X}) & \leftarrow \mathsf{pot\_chg}\_R_1.1(M_P,\bar{Y}),\ R_3(\bar{X}),\ \bar{X}[1]=\bar{Y}[1], \\
 & \quad M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{blk\_chg}\_R_1.1(M_P,\bar{Y}) & \leftarrow \mathsf{pot\_chg}\_R_1.1(M_P,\bar{Y}), \\
 & \quad \mathsf{blk\_prop}\_R_1.1 \rightsquigarrow R_3.1(M_C,\bar{X}), \\
 & \quad \bar{X}[1]=\bar{Y}[1],\ M_C = 1/M_P(\bar{Y})[1]. \\
\mathsf{blk\_prop}\_R_1.1 \rightsquigarrow R_3.1(M,\bar{X}) & \leftarrow \mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_3.1(M,\bar{X}), \\
 & \quad \mathsf{blk\_chg}\_R_3.1(M,\bar{X}).
\end{array}
$$

Analogously, from $R_4.(1,2) \rightarrow R_2.(1,2)$ `ON UPDATE OF PARENT CASCADE` (*Refd*), (MPC_1), (MPC_2):

$$
\begin{array}{ll}
\mathsf{new\_refd}\_R_2.(1,2)\_\mathsf{by}\_R_4.(1,2)(\bar{V}) & \leftarrow R_C(\bar{X}),\ \ \mathsf{chg}\_R_4.(1,2)(M,\bar{X}), \\
 & \quad M(\bar{X})[1,2]=\bar{V}. \\
\mathsf{prp}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M_C,\bar{X}) & \leftarrow \mathsf{chg}\_R_2.(1,2)(M_P,\bar{Y}),\ R_4(\bar{X}), \\
 & \quad \bar{X}[1,2]=\bar{Y}[1,2],\ M_C=(1,2)/ \\
 & \quad M_P(\bar{Y})[1,2]. \\
\mathsf{pot\_prp}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M_C,\bar{X}) & \leftarrow \mathsf{pot\_chg}\_R_2.(1,2)(M_P,\bar{Y}),\ R_4(\bar{X}), \\
 & \quad \bar{X}[1,2]=\bar{Y}[1,2],\ M_C=(1,2)/ \\
 & \quad M_P(\bar{Y})[1,2]. \\
\mathsf{blk\_chg}\_R_2.(1,2)(M_P,\bar{Y}) & \leftarrow \mathsf{pot\_chg}\_R_2.(1,2)(M_P,\bar{Y}), \\
 & \quad \mathsf{blk\_prop}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M_C,\bar{X}), \\
 & \quad \bar{X}[1,2]=\bar{Y}[1,2],\ M_C=(1,2)/ \\
 & \quad M_P(\bar{Y})[1,2]. \\
\mathsf{blk\_prop}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M,\bar{X}) & \leftarrow \mathsf{pot\_prp}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M,\bar{X}), \\
 & \quad \mathsf{blk\_chg}\_R_4.(1,2)(M,\bar{X}).
\end{array}
$$

Analogously, from $R_4.(1,3) \to R_3.(1,2)$ `ON UPDATE OF PARENT CASCADE` (*Refd*), (MPC_1), (MPC_2):

$$\begin{aligned}
&\mathsf{new_refd}_R_3.(1,2)_\mathsf{by}_R_4.(1,3)(\bar{V}) &&\leftarrow R_C(\bar{X}),\ \ \mathsf{chg}_R_4.(1,3)(M,\bar{X}),\\
&&&\quad M(\bar{X})[1,3] = \bar{V}.\\
&\mathsf{prp}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M_C,\bar{X}) &&\leftarrow \mathsf{chg}_R_3.(1,2)(M_P,\bar{Y}),\ R_4(\bar{X}),\\
&&&\quad \bar{X}[1,3] = \bar{Y}[1,2],\ M_C = (1,3)/\\
&&&\quad M_P(\bar{Y})[1,2].\\
&\mathsf{pot_prp}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M_C,\bar{X}) &&\leftarrow \mathsf{pot_chg}_R_3.(1,2)(M_P,\bar{Y}),\ R_4(\bar{X}),\\
&&&\quad \bar{X}[1,3] = \bar{Y}[1,2],\ M_C = (1,3)/\\
&&&\quad M_P(\bar{Y})[1,2].\\
&\mathsf{blk_chg}_R_3.(1,2)(M_P,\bar{Y}) &&\leftarrow \mathsf{pot_chg}_R_3.(1,2)(M_P,\bar{Y}),\\
&&&\quad \mathsf{blk_prop}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M_C,\bar{X}),\\
&&&\quad \bar{X}[1,3] = \bar{Y}[1,2],\ M_C = (1,3)/\\
&&&\quad M_P(\bar{Y})[1,2].\\
&\mathsf{blk_prop}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M,\bar{X}) &&\leftarrow \mathsf{pot_prp}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M,\bar{X}),\\
&&&\quad \mathsf{blk_chg}_R_4.(1,3)(M,\bar{X}).
\end{aligned}$$

For the (primary and foreign) key $R_1.1$ (CH_1) and (RCK), translate the external updates to R_1 into their effects on $R_1.1$ and do bookkeping of referenceable values:

$$\begin{aligned}
&\mathsf{pot_chg}_R_1.1(M,\bar{X}) &&\leftarrow \mathsf{pot_prp}_{\triangleright} \rightsquigarrow R_1.1(M',\bar{X}),\ M(\bar{X})[1] \neq \bar{X}[1].\\
&\mathsf{chg}_R_1.1(M,\bar{X}) &&\leftarrow \mathsf{prp}_{\triangleright} \rightsquigarrow R_1.1(M',\bar{X}),\ M(\bar{X})[1] \neq \bar{X}[1].\\
&\mathsf{new_refable}_R_1.1(\bar{V}) &&\leftarrow \mathsf{chg}_R_1.1(M,\bar{X}),\ M(\bar{X})[1] = \bar{V}.
\end{aligned}$$

The foreign key $R_2.1$ does not overlap with other foreign keys, thus (CH_1) considers only its "own" parent key (again we omit $\triangleright R_2$):

$$\begin{aligned}
&\mathsf{pot_chg}_R_2.1(M,\bar{X}) \leftarrow \mathsf{pot_prp}_R_1.1 \rightsquigarrow R_2.1(M',\bar{X}),\ M = M'[1],\ M(\bar{X})[1] \neq \bar{X}[1].\\
&\mathsf{chg}_R_2.1(M,\bar{X}) \leftarrow \mathsf{prp}_R_1.1 \rightsquigarrow R_2.1(M',\bar{X}),\ M = M'[1],\ M(\bar{X})[1] \neq \bar{X}[1].
\end{aligned}$$

Analogous for $R_3.1$.

For the key $R_2.(1,2)$ (CH_1) and (RCK), translate the incoming updates along $R_2.1 \to R_1.1$ `ON UPDATE OF PARENT CASCADE` to updates into their effects on $R_2.(1,2)$ and do bookkeping of referenceable values (for space restrictions, we omit the influence of $\triangleright R_2$ in (CH_1)):

$$\begin{aligned}
&\mathsf{pot_chg}_R_2.(1,2)(M,\bar{X}) &&\leftarrow \mathsf{pot_prp}_R_1.1 \rightsquigarrow R_2.1(M',\bar{X}),\\
&&&\quad M = M'[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].\\
&\mathsf{chg}_R_2.(1,2)(M,\bar{X}) &&\leftarrow \mathsf{prp}_R_1.1 \rightsquigarrow R_2.1(M',\bar{X}),\\
&&&\quad M = M'[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].\\
&\mathsf{new_refable}_R_2.(1,2)(\bar{V}) &&\leftarrow \mathsf{chg}_R_2.(1,2)(M,\bar{X}),\ M(\bar{X})[1,2] = \bar{V}.
\end{aligned}$$

Analogous for the key $R_3.(1,2)$.

The foreign key $R_4.(1,2)$ can be influenced either by its own parent key (which has to be regarded as atomic, thus, it cannot be augmented by any propagation along $R_4.(1,3) \to R_3.(1,2)$), or (if there is no change at the parent) by a propagation along $R_4.(1,3) \to R_3.(1,2)$ (again we omit $\triangleright R_4$). The rule schema (CH_1)

yields the following rules:

$$\begin{aligned}
\mathsf{pot_chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{pot_prp}_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M',\bar{X}),\\
& M = M_1[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].\\
\mathsf{chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{prp}_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M_1,\bar{X}),\\
& M = M_1[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].\\
\mathsf{pot_chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{pot_prp}_R_3.(1,2) \rightsquigarrow R_4.(1,2)(M',\bar{X}),\\
& M = M_1[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].\\
\mathsf{chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{prp}_R_3.(1,2) \rightsquigarrow R_4.(1,2)(M_1,\bar{X}),\\
& \neg\exists M_2 :\ \mathsf{prp}_R_2.(1,2) \rightsquigarrow R_4.(1,3)(M_2,\bar{X}),\\
& M = M_1[1,2],\ M(\bar{X})[1,2] \neq \bar{X}[1,2].
\end{aligned}$$

Analogous for $R_4.(1,3)$.
(MCN) contributes rules for $R_4.(1,2)$ and $R_4.(1,3)$ since there we have overlapping foreign keys which are changed by different parent key propagations:

$$\begin{aligned}
\mathsf{blk_chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{pot_chg}_R_4.(1,2)(M,\bar{X}),\\
& \mathsf{prp}_R_3.(1,2) \rightsquigarrow R_4.(1,3)(M',\bar{X}),\\
& M[1] = M'[1],\ \neg\,\mathsf{rem_refable}_R_2.(1,2)(M(\bar{X})[1,2]),\\
& \neg\,\mathsf{new_refable}_R_2.(1,2)(M(\bar{X})[1,2]).
\end{aligned}$$

$$\begin{aligned}
\mathsf{blk_chg}_R_4.(1,3)(M,\bar{X}) \leftarrow\ & \mathsf{pot_chg}_R_4.(1,3)(M,\bar{X}),\\
& \mathsf{prp}_R_2.(1,2) \rightsquigarrow R_4.(1,2)(M',\bar{X}),\\
& M[1] = M'[1],\ \neg\ \mathsf{rem_refable}_R_3.(1,3)(M(\bar{X})[1,3]),\\
& \neg\,\mathsf{new_refable}_R_3.(1,3)(M(\bar{X})[1,3]).
\end{aligned}$$

Finally, the schema (CH_2) adds the following rules for interferences between the overlapping foreign keys $R_4.(1,2)$ and $R_4.(1,3)$:

$$\begin{aligned}
\mathsf{allow_chg}_R_4.(1.2)\cap(1,3)(M,\bar{X}) \leftarrow\ & \\
& \mathsf{chg}_R_4.(1,2)(M_1,\bar{X}),\ \neg\mathsf{blk_chg}_R_4.(1,2)(M_1,\bar{X}),\ M = M_1[1],\\
& \mathsf{chg}_R_4.(1,3)(M_2,\bar{X}),\ \neg\mathsf{blk_chg}_R_4.(1,3)(M_2,\bar{X}),\ M = M_2[1].\\
\mathsf{blk_chg}_R_4.(1,2)(M,\bar{X}) \leftarrow\ & \mathsf{pot_chg}_R_4.(1,2)(M,\bar{X}),\\
& \neg\mathsf{allow_chg}_R.(1,2)\cap(1,3)(M',\bar{X}),\ M' = M[1].\\
\mathsf{blk_chg}_R_4.(1,3)(M,\bar{X}) \leftarrow\ & \mathsf{pot_chg}_R_4.(1,3)(M,\bar{X}),\\
& \neg\mathsf{allow_chg}_R.(1,2)\cap(1,3)(M',\bar{X}),\ M' = M[1].
\end{aligned}$$

Evaluating this program wrt. the well-founded semantics (via the AFP characterization) yields the following sequence of truth values (e.g., "tftf. . . " denoting "true-false-true-false-. . . "):

$\triangleright\mathsf{mod}\_R_1(1/n,(a,\ldots))$	tttt. . .
$\mathsf{pot\_chg}\_R_1.1(1/n,(a,\ldots))$	tttt. . .
$\mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_2.1(1/n,(a,b,\ldots))$	tttt. . .
$\mathsf{pot\_chg}\_R_2.(1,2)(1/n,(a,b,\ldots))$	tttt. . .
$\mathsf{pot\_prp}\_R_2.(1,2) \rightsquigarrow R_4.(1,2)(1/n,(a,b,c,\ldots))$	tttt. . .
$\mathsf{pot\_chg}\_R_4.(1,2)(1/n,(a,b,c,\ldots))$	tttt. . .
$\mathsf{pot\_prp}\_R_1.1 \rightsquigarrow R_3.1(1/n,(a,c,\ldots))$	tttt. . .
$\mathsf{pot\_chg}\_R_3.(1,2)(1/n,(a,c,\ldots))$	tttt. . .
$\mathsf{pot\_prp}\_R_3.(1,2) \rightsquigarrow R_4.(1,3)(1/n,(a,b,c,\ldots))$	tttt. . .
$\mathsf{pot\_chg}\_R_4.(1,3)(1/n,(a,b,c,\ldots))$	tttt. . .

mod_$R_1(1/n,(a,\ldots))$	tftf...
chg_$R_1.1(1/n,(a,\ldots))$	tftf...
prp_$R_1.1 \rightsquigarrow R_2.1(1/n,(a,b,\ldots))$	tftf...
chg_$R_2.(1,2)(1/n,(a,b,\ldots))$	tftf...
prp_$R_2.(1,2) \rightsquigarrow R_4.(1,2)(1/n,(a,b,c,\ldots))$	tftf...
chg_$R_4.(1,2)(1/n,(a,b,c,\ldots))$	tftf...
prp_$R_1.1 \rightsquigarrow R_3.1(1/n,(a,c,\ldots))$	tftf...
chg_$R_3.(1,2)(1/n,(a,c,\ldots))$	tftf...
prp_$R_3.(1,2) \rightsquigarrow R_4.(1,3)(1/n,(a,b,c,\ldots))$	tftf...
chg_$R_4.(1,3)(1/n,(a,b,c,\ldots))$	tftf...
new_refable_$R_1.1(n)$	tftf...
new_refable_$R_2.(1,2)(n,b)$	tftf...
new_refable_$R_3.(1,2)(n,c)$	tftf...
new_refd_$R_1.1$_by_$R_2.1(n)$	tftf...
new_refd_$R_2.(1,2)$_by_$R_4.(1,2)(n,b)$	tftf...
new_refd_$R_3.(1,2)$_by_$R_4.(1,3)(n,c)$	tftf...
blk_chg_$R_1.1(1/n,(a,\ldots))$	tftf...
blk_prop_$R_1.1 \rightsquigarrow R_2.1(1/n,(a,b,\ldots))$	tftf...
blk_chg_$R_2.(1,2)(1/n,(a,b,\ldots))$	tftf...
blk_prop_$R_2.(1,2) \rightsquigarrow R_4.(1,2)(1/n,(a,b,c,\ldots))$	tftf...
blk_chg_$R_4.(1,2)(1/n,(a,b,c,\ldots))$	tftf...
blk_prop_$R_1.1 \rightsquigarrow R_3.1(1/n,(a,c,\ldots))$	tftf...
blk_chg_$R_3.(1,2)(1/n,(a,c,\ldots))$	tftf...
blk_prop_$R_3.(1,2) \rightsquigarrow R_4.(1,3)(1/n,(a,b,c,\ldots))$	tftf...
blk_chg_$R_4.(1,3)(1/n,(a,b,c,\ldots))$	tftf...
allow_chg_$R_4.(1.2) \cap (1,3)\,(1/n,(a,b,c,\ldots))$	tftf...
blk_chg_$R_4.(1,2)(1/n,(a,b,c,\ldots))$	tftf...
blk_chg_$R_4.(1,3)(1/n,(a,b,c,\ldots))$	tftf...

Thus, all actual modifications and changes, as well as all blockings are undefined in the well-founded model. Nevertheless, the modification is admissible.

B. GAME-THEORETIC CHARACTERIZATION OF REFERENTIAL ACTIONS WITH UPDATES

In this section, we present an equivalent game-theoretic characterization of maximal admissible sets of updates. Here, we need to consider also a *history* of the game which was not required for the simpler game-theoretic characterization in Section 3.3 with deletions only (which was in the famous *win-move*-style).

For given $U_{\triangleright}$, Player I claims a subset $U \subset U_{\triangleright}$ to be maximal and admissible. In her first move, Player II chooses to falsify either the maximality or the admissibility. If Player II challenges the maximality, she chooses a proper superset U' such that $U \subsetneq U' \subset U_{\triangleright}$ which she claims to be maximal and admissible, then the roles are changed. Thus, after finitely many moves, a player challenges the admissibility of a set U suggested by the other player by examining this set with respect to its coherence and feasibility by questions.

The other player has to defend U to be admissible by stepwise showing what updates are actually executed. By doing this, he constructs $\Delta(U)$. The game is an abstraction of the logic programming characterization in the sense that I uses only non-subsumed updates (anticipating the overall result), thus the details of interfering updates can be ignored. This abstraction step is similar to that in Section 5 for deriving the practical results from the construction of the well-founded and stable models.

B.1 Rules of the Game

B.1.1 *Setting and Initialization*

Setting. The positions of the game are all tuples of the database D: $\boxed{R(\bar{x})}$ such that $R(\bar{x}) \in D$ and $|\{\triangleright_{\mathsf{ins}}\_R(\bar{X}) \text{ such that } \triangleright_{\mathsf{ins}}\_R(\bar{X}) \in U_{\triangleright}\}|$ many positions $\boxed{\varepsilon}$. Thus, the "board" is practically a graphical representation of the database (see Example 15 and Figures 6–8). The game is played by putting *plates* that represent the update operations performed on the database: Each plate consists of a source tuple ($\in D$), an update (over the active domain of the database and the updates; $\mathsf{adom}(D) \cup \mathsf{adom}(U_{\triangleright})$), and a result tuple, for example, $\boxed{R(a,b,c)|\ \mathsf{mod}\_R([1/x, 2/y], (a,b,c))|R(x,y,c)}$. The update plates also contain positions $\square$ (as many boxes as there are atomic updates described by the plate), indicating how many cascading steps are needed for founding the update. Each position can be filled with a pebble (as a question) or with a number. For example, for the above plate, there are two positions (for $[1/x]$ and for $[2/y]$): $\boxed{R(a,b,c)|\ \mathsf{mod}\_R([1/x, 2/y], (a,b,c))\ \langle 2, \bullet\rangle|R(x,y,c)}$ means that the modification of position 1 to x is founded by two cascading steps, and II currently forces I to tell her how many steps are needed for founding the modification $[2/y]$. There are four kinds of update plates:

No change:	$\boxed{R(\bar{x})\|\mathsf{unchanged}\|R(\bar{x})}$,
Deletion:	$\boxed{R(\bar{x})\|\mathsf{del}\_R(\bar{x})\ \langle\square\rangle\|\varepsilon}$,
Insertion:	$\boxed{\varepsilon\|\ \mathsf{ins}\_R(\bar{x})\ \langle\square\rangle\|R(\bar{x})}$,
Modification:	$\boxed{R(\bar{x})\|\mathsf{mod}\_R([m_1,...,m_n],\bar{x})\ \langle\square^n\rangle\|R(M(\bar{x}))}$; $M = [m_1,...,m_n]$.

Player I: Start. Player I claims that a set $U \subset U_{\triangleright}$ of updates is maximal admissible and puts the following plates with numbers on suitable positions:

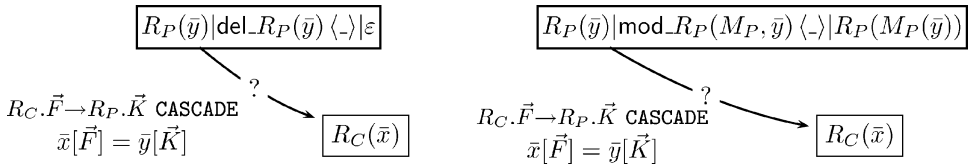
$\boxed{R(\bar{x})\|\mathsf{del}\_R(\bar{x})\ \langle 0\rangle\|\varepsilon}$	: $\triangleright\mathsf{del}\_R(\bar{x}) \in U$
$\boxed{R(\bar{x})\|\mathsf{mod}\_R([m_1,...,m_n],\bar{x})\ \langle 0^n\rangle\|R(M(\bar{x}))}$	: $\triangleright\mathsf{mod}\_R(M,\bar{x}) \in U$; $M = [m_1,\ldots,m_n]$
$\boxed{\varepsilon\|\mathsf{ins}\_R(\bar{x})\ \langle 0\rangle\|R(\bar{x})}$	: $\triangleright_{\mathsf{ins}}\_R(\bar{x}) \in U$

B.1.2 *Questions and Answers.* The game is played such that Player II asks "questions", attacking the claim of Player I: II attacks the admissibility

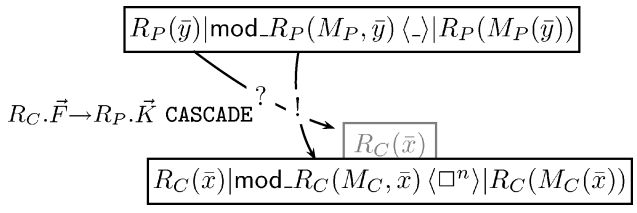
(cf. Definition 4.1) of U, that is, foundedness, completeness, and feasibility. Coherence is inherent to the game, and uniqueness of key values is guaranteed by the winning conditions. I has to answer each attack—if he has no answer, he loses. We will describe each aspect below by *attack-defense*-pairs: II asks a question by pointing to an instance of one of the above aspects, and I has to show how to guarantee the respective property. In most answers, I will put a new plate (thereby constructing $\Delta(U)$); then the number positions of the plate are initially empty.

Completeness / Cascading. If an update plate is positioned on a tuple that has a child tuple with a CASCADE reference, II can ask I to cascade the update (e.g., when trying to follow a $\mathcal{DC}^* \circ \mathcal{DR}$ or $\mathcal{DC}^* \circ \mathcal{DN}$ chain to a *problem situation*). II answers by materializing the cascaded update:

Cascade Attack. For a deletion plate $\boxed{R_P(\bar{y})|\mathsf{del\_}R_P(\bar{y})\,\langle\_\rangle|\varepsilon}$ or a modification plate $\boxed{R_P(\bar{y})|\mathsf{mod\_}R_P(M_P,\,\bar{y})\,\langle\_\rangle|\ldots}$, a rac $R_C.\vec{F} \to R_P.\vec{K}$ ON DELETE/UPDATE OF PARENT CASCADE, and a tuple $R_C(\bar{x})$ s.t. $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$ and—in case of a modification—$\bar{y}[\vec{K}] \neq M_P(\bar{y})[\vec{K}]$, Player II can put a *propagate-wire* labelled with the *ric* from the updated parent tuple to the referencing tuple, asking "what about this reference?":



Cascade Answer. A *propagate-wire* from a plate $\boxed{R_P(\bar{y})|\mathsf{del\_}R_P(\bar{y})\,\langle\_\rangle|\varepsilon}$ or $\boxed{R_P(\bar{y})|\mathsf{mod\_}R_P(M_P,\,\bar{y})\,\langle\_\rangle|\ldots}$ to a tuple $R_C(\bar{x})$ is answered by putting a plate $\boxed{R_C(\bar{x})|\mathsf{del\_}R_C(\bar{x})\,\langle\square\rangle|\varepsilon}$ respectively $\boxed{R_C(\bar{x})|\mathsf{mod\_}R_C(M_C,\bar{x})\,\langle\square^n\rangle|\ldots}$ onto $R_C(\bar{x})$, and putting an action wire labelled with $R_C.\vec{F} \to R_P.\vec{K}$ from the update component of the parent to the update component of the child.

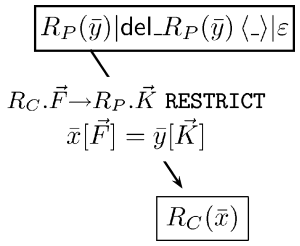


In case of a modification, the values of the corresponding candidate keys and foreign keys must be consistent, that is, $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$.

If the target tuple already holds a plate, there is nothing more to do than adding an action wire from the update component of the parent to the update component of the child. Player I loses if the plates are inconsistent (i.e., the child plate does not represent the correct key value—trapped).

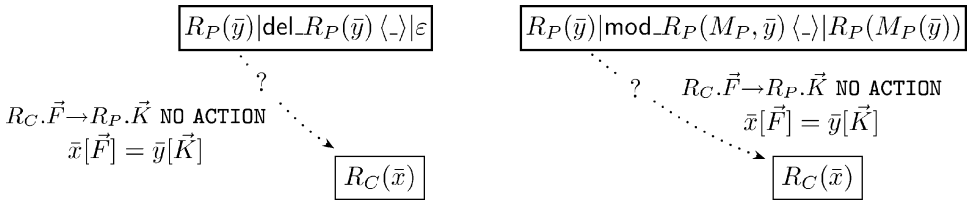
Feasibility: Restricting. If an update plate is positioned on a tuple that has a child tuple with a RESTRICT reference, II can show the reference and wins immediately:

Restrict Attack. For a deletion plate $\boxed{R_P(\bar{y})|\mathsf{del}\_R_P(\bar{y})\,\langle\_\rangle|\varepsilon}$ or a modification plate $\boxed{R_P(\bar{y})|\mathsf{mod}\_R_P(M,\bar{y})\,\langle\_\rangle|\ldots}$, a *rac* $R_C.\vec{F} \to R_P.\vec{K}$ ON DELETE/UPDATE OF PARENT RESTRICT, and a tuple $R_C(\bar{x})$ s.t. $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$, Player II can point to this tuple, and Player I immediately loses.



Feasibility: No Action. If an update plate is positioned on a tuple that has a child tuple with a NO ACTION reference, II can show the reference, asking what happens to that child/reference. I answers by showing (claiming) that the child is deleted or modified (which must be founded from somewhere else and which must be admissible, leading to II's next move):

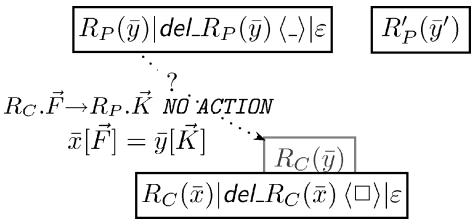
No Action Attack. For a deletion plate $\boxed{R_P(\bar{y})|\mathsf{del}\_R_P(\bar{y})\,\langle\_\rangle|\varepsilon}$ or a modification plate $\boxed{R_P(\bar{y})|\mathsf{mod}\_R_P(M_P,\bar{y})\,\langle\_\rangle|\ldots}$, a *rac* $R_C.\vec{F} \to R_P.\vec{K}$ ON DELETE/UPDATE OF PARENT NO ACTION, and a tuple $R_C(\bar{x})$ such that $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$ and—in case of a modification—$\bar{y}[\vec{K}] \neq M_P(\bar{y})[\vec{K}]$, Player II can put a *no-action-wire* labelled with the *ric* from the source tuple of the update to the referencing tuple, asking "what about this reference?" (the deletion situation on the left side will be continued in Example 13 with the answer and a following *Founding Attack* step):



No Action Answer. For showing how the referencing tuple is adapted, Player II puts some (consistent) update plate on the result tuple.

Such situations frequently occur in a diamond, where the update cascades along another way (cf. the Example 15 that describes a complete game).

Example 13 *Update Game—Selected Steps.*
Continuing the above situation (left side; deletion), I puts a deletion plate on the referencing tuple, arguing that this tuple will also be deleted. Note that this "deletion" is at that time just a claim, and its *foundedness* will very probably be attacked by II (as will be shown below).

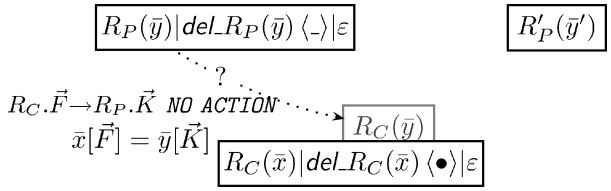


In case that the referencing tuple already holds a plate, there is nothing to do. In this case, player I loses if the plates are inconsistent (i.e., the child plate does not represent the correct key value—again trapped).

Founding. If I answers—for example as described above—with putting a delete or update plate somewhere, II can ask him how this modification is founded (or, if it is an update plate where several foreign keys change, how each component of it is founded). I has to answer by showing a parent that cascades an appropriate modification, and by saying how many steps are needed to prove the correctness.

Founding Attack. For any plate with some empty number position, $\boxed{R(\bar{x})|\mathsf{del\_}R(\bar{x})\,\langle\square\rangle|\varepsilon}$ or $\boxed{R(\bar{x})|\mathsf{mod\_}R([m_1,\ldots,m_n],\bar{x})\,\langle k_1,\ldots,\square,\ldots,k_n\rangle|\ldots}$, Player II can place a pebble on the number position, asking "why m_i?": $\boxed{R(\bar{x})|\mathsf{del\_}R(\bar{x})\,\langle\bullet\rangle|\varepsilon}$ or $\boxed{R(\bar{x})|\mathsf{mod\_}R([m_1,...,m_n],\bar{x})\,\langle k_1,...,k_{i-1},\bullet,k_{i+1},...,k_n\rangle|\ldots}$.

Example 13 *Update Game—Selected Steps (Cont'd).* Continuing the above situation, II will ask how the newly placed deletion plate is founded (by putting a pebble on the empty "□" on the plate).

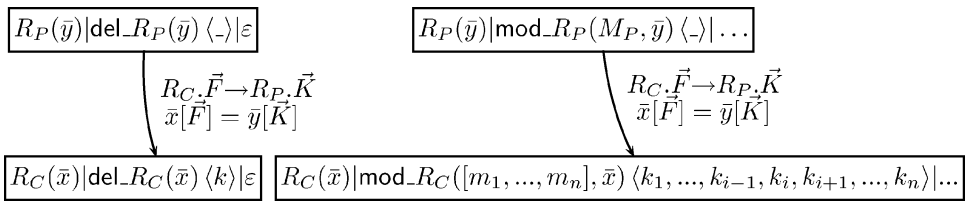


Founding Answer. For a new pebble, $\boxed{R_C(\bar{x})|\mathsf{del\_}R_C(\bar{x})\,\langle\bullet\rangle|\varepsilon}$ respectively $\boxed{R_C(\bar{x})|\mathsf{mod\_}R_C([m_1,...,m_n],\bar{x})\,\langle k_1,...,k_{i-1},\bullet,k_{i+1},...,k_n\rangle|\ldots}$, I chooses a suitable plate for founding the questioned update (this can be either a plate that is already present, or I puts a new plate):

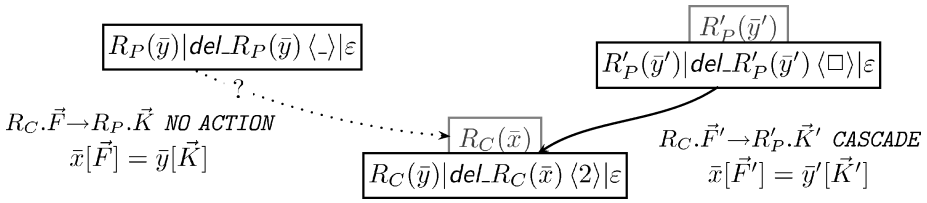
—in case of a deletion: $\boxed{R_P(\bar{y})|\mathsf{del\_}R_P(\bar{y})\,\langle\_\rangle|\varepsilon}$ such that $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$, or

—in case of a modification: $\boxed{R_P(\bar{y})|\mathsf{mod\_}R_P(M_P,\bar{y})\,\langle\_\rangle|\ldots}$ such that $\bar{x}[\vec{F}] = \bar{y}[\vec{K}]$, $[m_1,...,m_n][\vec{F}] = \vec{F}/M_P(\bar{y})[\vec{K}]$, and $m_i \in \vec{F}/M_P(\bar{y})[\vec{K}]$).

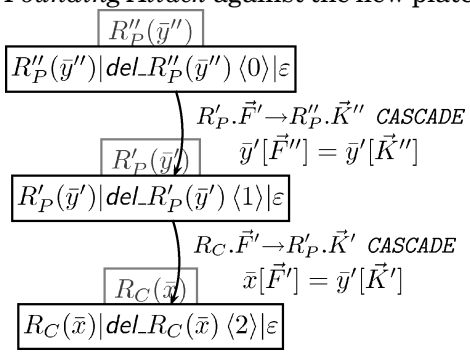
Then, I adds an *action-wire* marked with a *ric* $R_C.\vec{F} \rightarrow R_P.\vec{K}$ from the update component of the founding plate to the update component $\mathsf{del\_}R_C(\bar{x})$ respectively $\mathsf{mod\_}R_C([m_1\ldots,m_n],\bar{x})\,\langle k_1,\ldots,k_{i-1},k_i,k_{i+1},\ldots,k_n\rangle$ of the plate under consideration and replaces the pebble by some $k_i \in \mathbb{N}$ (claiming that the update is founded in k_i steps):



Example 13 *Update Game—Selected Steps* (*Cont'd*). Consider again the above situation. Player I answers the pebble by showing another parent tuple (now, $R'_P(\bar{y}')$ comes actually into play) of the referencing tuple from which the deletion cascades, and telling how many steps are required from a founding update:



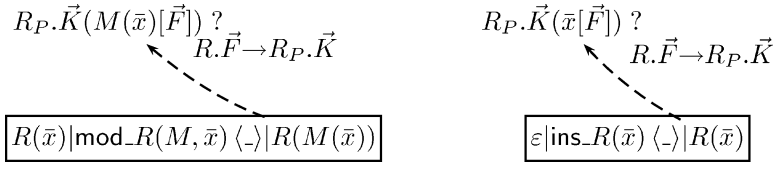
The next move by II is probably again a *Founding Attack* against the new plate. This is continued until I's answers reach a founding external update (showing only the right side of the example board): I places the pebble on the update plate at $R'_P(\bar{y}')$, II replaces it by a "1" and links it to the founding update plate for an external update, $\mathsf{del\_}R''_P(\bar{y}'')$ (which has been put in the initialization).



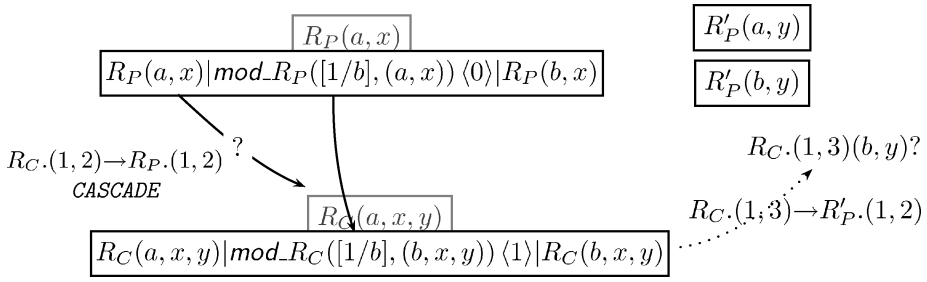
Feasibility: Referenced Parents Needed. Feasibility of an update is not only concerned with the children of the tuple (as handled above by *Restrict Attack* and *No Action Attack*), but has also to consider updates of foreign keys, searching for parents.

If I answers with putting an update plate somewhere that changes a foreign key, II can ask him what parent is referenced (note that this is different from asking how the update is founded). I has to answer by showing a parent that provides an appropriate key value (this can either be an unchanged tuple of the original database, or the result of a modification or insertion).

Referencing Attack. For a modification $R(\bar{x})|\mathsf{mod\_}R(M,\bar{x})\ \langle\_\rangle|\ldots$ or insertion $\varepsilon|\mathsf{ins\_}R(\bar{x})\ \langle\_\rangle|R(\bar{x})$ and a *ric* $R.\vec{F}\to R_P.\vec{K}$, Player II can put a *reference-wire* with a loose end, labelled with the *ric*, on the result entry, asking "referencing what?":

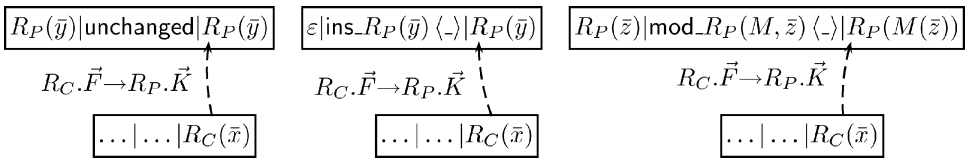


Example 14 *Update Game: Referencing.* A *Reference Attack* is for example played if a tuple is "moved" from one parent to another: Consider relations R_C, R_P and R'_P with *ric*'s $R_C.(1,2) \rightarrow R_P.(1,2)$ `CASCADE` and $R_C.(1,3) \rightarrow R'_P.(1,2)$ `NO ACTION` where the foreign keys overlap. Let D contain the tuples $R_P(a,x)$, $R'_P(a,y)$, $R'_P(b,y)$, $R_C(a,x,y)$. Then, the modification $\triangleright \mathsf{mod\_}R_P([1/b],(a,x))$ cascades to $\mathsf{mod\_}R_C([1/b],(a,x,y))$, resulting in the tuple $R_C(b,x,y)$ that now references $R'_P(b,y)$. The game starts with the database and the plate $\boxed{R_P(a,x)|\mathsf{mod\_}R_P([1/b],(a,x))\,\langle 0\rangle|R_P(b,x)}$. After playing *Cascade Attack* and *Cascade Answer*, the situation looks as depicted below. In this situation, Player II—knowing that the reference from $R'_P(a,y)$ breaks—plays *Reference Attack* for the reference $R_C.(1,3) \rightarrow R'_P.(1,2)$:



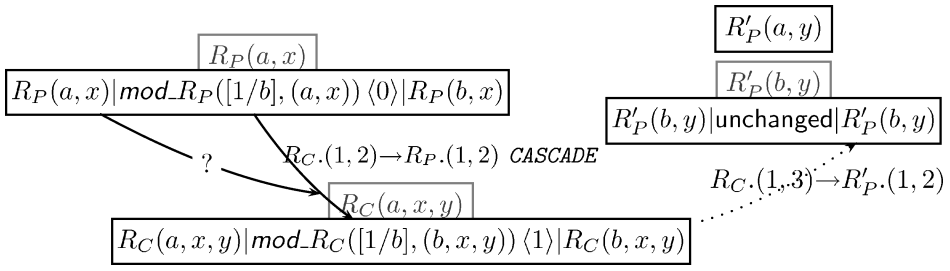
Player `I` has then to find a target for the open reference.

Referencing Answer. A dangling reference wire starting in a plate $\boxed{\ldots|\ldots|R_C(\bar{x})}$ labelled with a *ric* $R_C.\vec{F} \rightarrow R_P.\vec{K}$ is answered by connecting it to a plate (which also can be positioned in this move on a tuple not holding a plate) such that the result tuple provides the referenced key value $R_P(\bar{y})$, i.e., $\bar{y}[\vec{K}] = \bar{x}[\vec{F}]$:



In case that `I` puts a new plate, II can again attack its founding and admissibility.

Example 14 *Update Game: Referencing* (*Cont'd*). In the above situation, `I` takes the open end, puts an unchanged plate on $R'_P(b,y)$ and connects the open end to its result:



Key Condition. If I answers by placing an update or insert plate whose key value does already exist, II can ask him how to retain the uniqueness of the key. I can answer by deleting or modifying the other tuple.

Key Attack. Player II can put an empty plate $\boxed{R(\bar{x})|?|?}$ on a database tuple $R(\bar{x})$ s.t. $\vec{K}$ is a key of R and there exists an insertion or modification plate with a result tuple $R(\bar{y})$ such that $\bar{x}[\vec{K}] = \bar{y}[\vec{K}]$ asking "what will happen to this tuple?".

Key Answer. Player I has to replace/fill the empty plate $\boxed{R(\bar{x})|?|?}$ by a delete or update plate (which then has to change the key value).

Maximality. Player II has another way to refute I's claim that his proposed set is maximal admissible: with the above attacks, only admissibility was checked. In her first move, II can also claim, that U is not maximal by choosing a subset U' such that $U \subsetneq U' \subset U_{\triangleright}$. Then, the roles are changed: I now asks questions in order to prove that U' is not admissible (or still not maximal).

B.1.3 *Winning Conditions and Termination.* Some winning conditions have already been mentioned above, for example, when II traps I by showing a restricting reference, or when I cheats by putting inconsistent plates, or when he has no defending answer. The more intricate situations have to do with the number of steps to justify an update. Since the game uses a "history", there are no infinite cycles (that lead to drawn positions in the delete-game).
For $\mathbb{N}_0 \cup \{\bullet\}$ let $<_\bullet$ be the complete ordering $<_{\mathbb{N}} \cup \{(\bullet, n) \mid n \in \mathbb{N}_0\}$.
For every deletion plate $P_C = \boxed{R_C(\bar{x})|\mathsf{del\_}R_C(\bar{x})\,\langle k\rangle|\varepsilon}$, let

$k_C := 1+\min_\bullet\{k_P \mid$ there is a plate $P_P = \boxed{R_P(\bar{y})|\mathsf{del\_}R_P(\bar{y})\,\langle k_P\rangle|\varepsilon}$ and an action wire labeled with a *ric* $R_C.\vec{F}\rightarrow R_P.\vec{K}$ ON DELETE CASCADE from P_P to $C\}$.

For every modification plate $C = \boxed{R_C(\bar{x})|\mathsf{mod\_}R_C([c_1,...,c_n],\bar{x})\,\langle k_1,...,k_n\rangle|\ldots}$, let $k_{C_i} := 1+\min_\bullet\{l_j \mid$ there is a $P_P = \boxed{R_P(\bar{y})|\mathsf{mod\_}R_P([p_1,\ldots,p_m],\bar{y})\,\langle l_1,\ldots,l_m\rangle|\ldots}$ and an action wire labeled with a *ric* $R_C.\vec{F} \rightarrow$ ON UPDATE OF PARENT CASCADE from P to C, $c_i \in \vec{F}$ and $p_j \in \vec{K}\}$.

A situation is won for Player II if one of the following conditions holds:

— Player I cannot answer.
— Player II shows a restriction (cf. *Restrict Attack*).
— There are two plates with the same key value.
— Player I "lies":
  — either, when an action wire connects two plates of different types, or when the numbers of founding steps are inconsistent:
  — There is an action wire connecting two deletion plates $\boxed{R_P(\bar{x})|\mathsf{del\_}R_P(\bar{x})\,\langle k_P\rangle|\varepsilon}$ and $\boxed{R_C(\bar{y})|\mathsf{del\_}R_C(\bar{y})\,\langle k_C\rangle|\varepsilon}$ such that $k_C \leq k_P$.
  — An action wire labeled with a *ric* $R_C.\vec{F} \to R_P.\vec{K}$ connects two modification plates $\boxed{R_P(\bar{x})|\mathsf{mod\_}R_P(M_P, \bar{x})\,\langle \bar{k}_P\rangle|\ldots}$ and $\boxed{R_C(\bar{y})|\mathsf{mod\_}R_C(M_C, \bar{y})\,\langle \bar{k}_C\rangle|\ldots}$ such that $M_C \supseteq \vec{F}/M_P(\bar{x})[\vec{K}]$ and $k_C(\vec{F}(i)) \leq k_P(\vec{K}(i))$ $(i \in \{1, \ldots, |\vec{F}|\})$.
  — An action wire labeled with a *ric* $R_C.\vec{F} \to R_P.\vec{K}$ connects two modification plates $\boxed{R_P(\bar{x})|\mathsf{mod\_}R_P(M_P, \bar{x})\,\langle \_\rangle|\ldots}$ and $\boxed{R_C(\bar{y})|\mathsf{mod\_}R_C(M_C, \bar{y})\,\langle \_\rangle|\ldots}$ such that $M_C \not\supseteq \vec{F}/M_P(\bar{x})[\vec{K}]$.
  (Note that this is enforced in a regular game by the rules *Cascade Answer* and *Referencing Answer*).
  — A no-action wire labeled with a *ric* $R_C.\vec{F} \to R_P.\vec{K}$ goes from a parent plate $\boxed{R_P(\bar{y})|\_|\_}$ to a child plate $\boxed{R_C(\bar{x})|\_|R_C(\bar{x}')}$ and $\bar{x}'[\vec{F}] = \bar{y}\,[\vec{K}]$.

A situation is won for Player I if Player II has no more questions. Obviously, the game is finite since there are only finitely many positions, where a plate can be placed and every plate has only finitely many $\square$ positions.

Definition B.1. A starting set U is *won* for Player I iff he has a winning strategy: no matter how Player II moves, Player I can win the game.

B.2 Example

The following example illustrates the game-theoretic characterization by playing a complete game for a given situation.

Example 15 *Update Game.* Consider again Example 7 with the database with racs as given in Figure 2 (using only $R_1, \ldots R_5$), where the `ON UPDATE` rac is the same as the rac given for `ON DELETE`. Consider the user request $U_\triangleright = \{\triangleright\mathsf{del\_}R_1(a), \triangleright\mathsf{mod\_}R_1(1/c, b)\}$.

The first moves are described in Figure 6: I claims that $U_\triangleright$ is admissible. Thus, the initialization consists of placing the corresponding plates $\boxed{R_1(a)|\mathsf{del\_}R_1(a)\,\langle 0\rangle|\varepsilon}$ and $\boxed{R_1(b)|\mathsf{mod\_}R_1(1/c, b)\,\langle 0\rangle|R_1(c)}$ at $R_1(a)$ and $R_1(b)$, respectively.

Player II first challenges the deletion (targeting to the `NO ACTION` reference from R_4 to R_3) by Cascade Attack and puts an action wire from $\boxed{R_1(a)|\mathsf{del\_}R_1(a)\,\langle\square\rangle|\varepsilon}$ to $R_3(a, y)$. I answers by Cascade Answer, putting

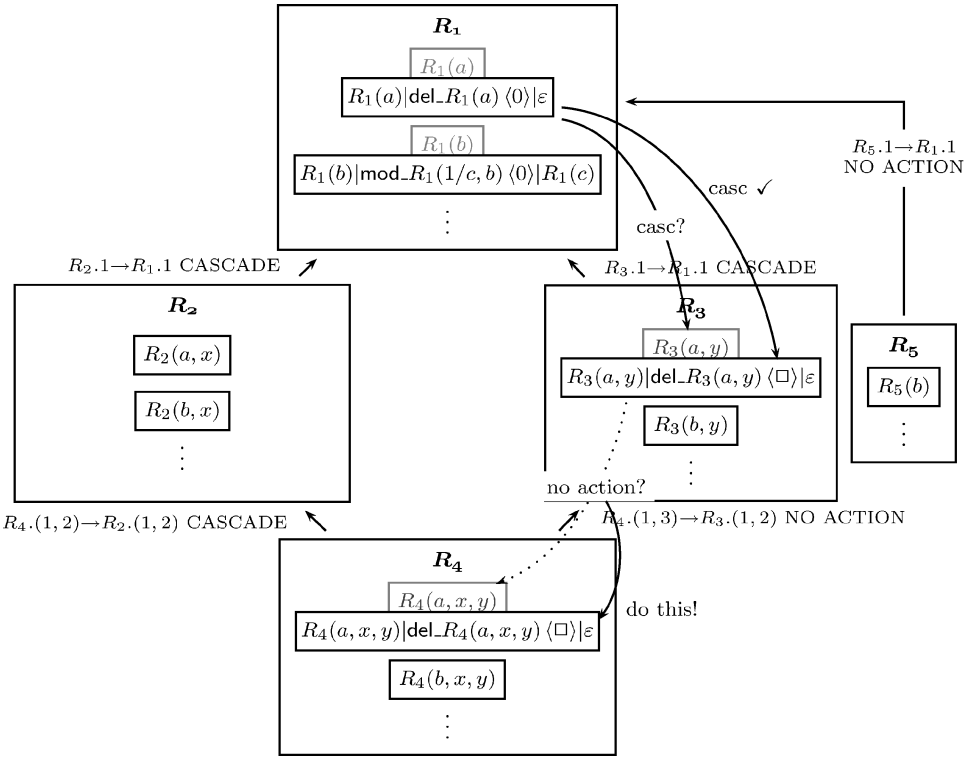


Fig. 6. Update Game: intermediate situation, fighting the "a"-deletion

$\boxed{R_3(a,\,y)|\mathsf{del\_}R_3(a,\,y)\,\langle\square\rangle|\varepsilon}$ on $R_3(a,\,y)$ and adding the action wire from the first plate to the second one.

Now, Player II uses the `NO ACTION` child $R_4(a, x,\, y)$ for a No Action Attack and puts a wire from $\boxed{R_3(a,\,y)|\mathsf{del\_}R_3(a,\,y)\,\langle\square\rangle|\varepsilon}$ to $R_4(a, x,\, y)$. Player I answers by putting a deletion plate $\boxed{R_4(a, x,\, y)|\mathsf{del\_}R_4(a, x,\, y)\,\langle\square\rangle|\varepsilon}$ on $R_4(a, x,\, y)$ (cf. Figure 6). Continue with Figure 7 with the next moves.

Player II now asks "why can you do this" by Founding Attack, placing a pebble on the $\square$ position of the plate that now looks like $\boxed{R_4(a, x,\, y)|\mathsf{del\_}R_4(a, x,\, y)\,\langle\bullet\rangle|\varepsilon}$. I replaces the pebble by a "2" and puts a $\boxed{R_2(a, x)|\mathsf{del\_}R_2(a, x)\,\langle\square\rangle|\varepsilon}$ plate on $R_2(a, x)$ and connects them by a `CASCADE` wire (action from R_2 to R_4).

Again, Player II applies Founding Attack, placing a pebble on the $\square$ position of that plate, yielding $\boxed{R_2(a, x)|\mathsf{del\_}R_2(a, x)\,\langle\bullet\rangle|\varepsilon}$. Now, I replaces the pebble by a "1" and connects the plate by another action wire from $\boxed{R_1(a)|\mathsf{del\_}R_1(a)\,\langle 0\rangle|\varepsilon}$ (that has been placed in the initialization).

As an intermediate result, I has now won the deletion of $R_1(a)$ since II has no more questions. Player I has generated all internal updates that are necessary to execute $\triangleright\mathsf{del\_}R_1(a)$ (cf. Figure 7).

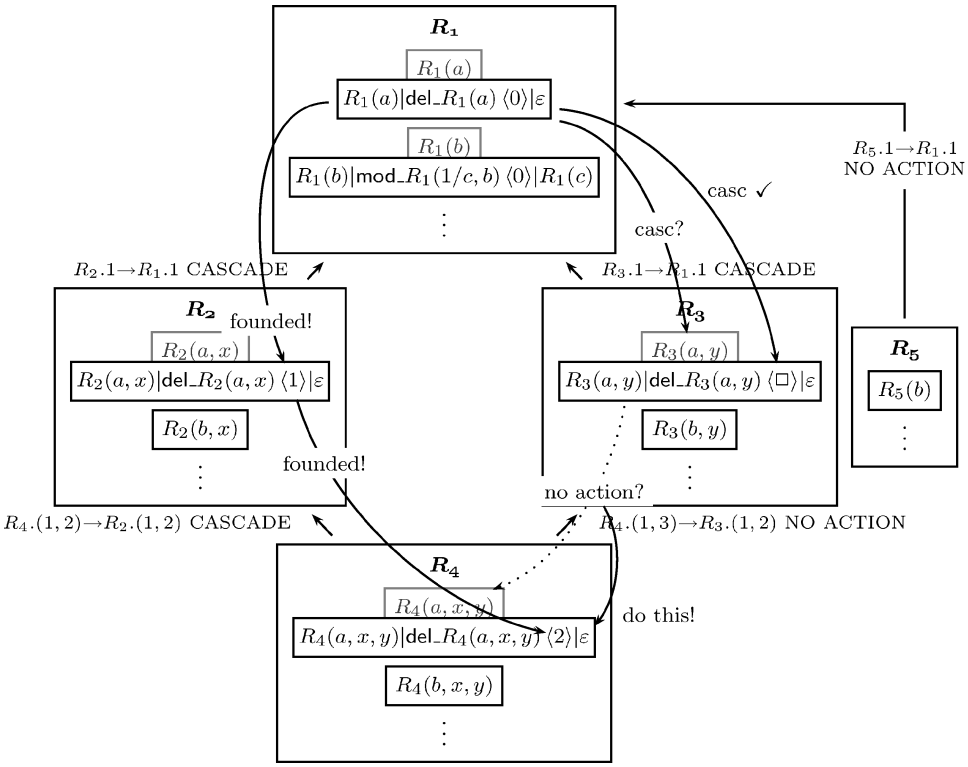


Fig. 7. Update Game: Situation after I won the "a"-deletion.

An analogous sequence of moves can for example be played for showing that I wins the modification of $R_1(b)$ for R_2, R_3, and R_4—but II already knows this so she doesn't play it.

Instead, Player II challenges then the modification of $R_1(b)$ with a No Action Attack by putting a wire from $\boxed{R_1(b)|\mathsf{mod\_}R_1(1/c,b)\ \langle 0\rangle|R_1(c)}$ to $R_5(b)$. She has not yet won! Player I can still answer by putting a $\boxed{R_5(b)|\mathsf{mod\_}R_5(1/c,b)\ \langle\square\rangle|R_5(c)}$ on $R_5(b)$. This move is completely legal. But now, Player II applies Founding Attack, placing a pebble on the $\square$ position of that plate. Then, Player I has no answer (since this update is unfounded) and loses (see Figure 8).

B.3 Equivalence

For a game on given D, $U_{\triangleright}$, and RA with a starting set U which is played until Player I wins, let

$$\Lambda := \{upd \mid \exists R(\bar{x}) : \boxed{R(\bar{x})|upd|\ldots} \text{ is played}\} \quad \text{and}$$
$$D' := \{R'(\bar{x}') \mid \exists R(\bar{x}) : \boxed{R(\bar{x})|\ldots|R'(\bar{x}')} \text{ is played}\} \cup$$
$$\cup \{R(\bar{x}) \mid R(\bar{x}) \in D \text{ and there is no plate on position } \boxed{R(\bar{x})}\}\ .$$

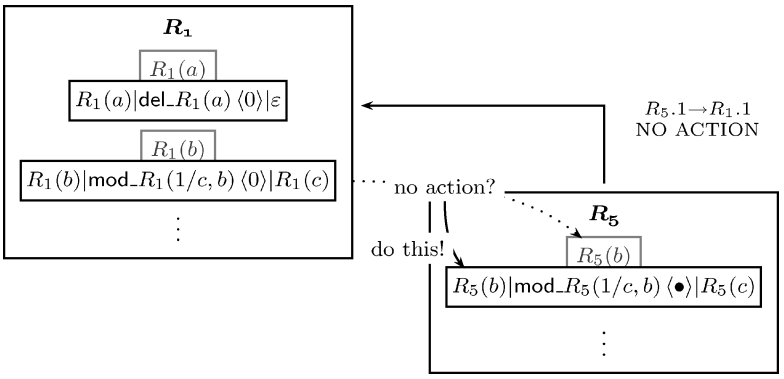


Fig. 8. Update Game: Excerpt of the situation after I lost the "b"-modification in R_5.

As mentioned above, the main difference between the game and the logic programming characterization is that only non-subsumed updates are "played", corresponding to already guessing a ($<_a$-maximal) stable model and showing its admissibility.

THEOREM B.2. *If a game is won for Player I,* $\Lambda = \Delta(U)$.

PROOF. The starting situation guarantees that $U \subset \Lambda$. Completeness is guaranteed by the "cascade" question, and minimality of Λ with respect to U is guaranteed by the "founding" question. □

COROLLARY B.3. $D' = D \pm \Lambda$ *is the database obtained by executing* U *on* D.

The final theorem states that the game characterizes exactly the $<_a$-maximal, that is, maximal admissible subsets:

THEOREM B.4. *A starting set* $U \subset U_\triangleright$ *is won for Player I, if and only if* U *is maximal with respect to* $U_\triangleright$ *and admissible.*

PROOF. For every property, Player II can ask questions:
Maximality: If U is not maximal, Player II choses a maximal admissible superset and wins his game.
The above lemma proved that $\Lambda = \Delta(U)$.
Λ satisfies conditions (1) and (2) of feasibility because otherwise Player II would have won by *Restrict Attack* (if she finds a restriction, Player II shows it and wins) or *No Action Attack* (Player I has to show what to do with the child tuple such that it no longer references the parent key value).
Λ satisfies conditions (3) and (4) of feasibility because otherwise Player II would have won by "referencing" (Player I has to show which tuple is referenced).
Λ is coherent since Player I can only place one plate on each tuple. Λ is key-preserving since otherwise Player II would have won by "key condition". □