An Adaptation Framework for View-based Data Sharing in Bipartite Network of Bidirectional Transformations

Soichiro Hidaka　Hiroyuki Kato　Masato Takeichi

In distributed environments, controlled data sharing is crucial for meeting security and other policy requirements. Dejima architecture supports such data sharing by compositions of bidirectional transformations (BX). It organizes bipartite networks of BX of which source databases in peers form the first set of nodes and the views form the second set of nodes, while each BX connects one of the first set as a source database and another one of the second set as a view. Sources and views form (multi) spans and cospans of BX, respectively. Such individual topology is also well studied. However, the network is relatively static in the sense that various reorganizations, including participation of new peers and changing conditions on data sharing, cannot be easily supported. In this presentation, we propose how to cope with such adaptations while maintaining the bipartite network structure, borrowing the type system from relational lenses that are known as BX on relations.

1 Introduction

In distributed environments, controlled data sharing is crucial for meeting security and other policy requirements. Dejima architecture [11] supports such data sharing by compositions of bidirectional transformations [5] [10] [2] [1]. Among variations in bidirectional transformations, asymmetric state-based approach is used in our framework. A bidirectional transformation is a pair of forward transformation function $\mathsf{get} : S \to V$ from source of type S to view of type V, and putback function $\mathsf{put} : S \times V \to S$ that takes a pair of original source of type S and updated view of type V to produce updated source of type S. In this paper, we follow the notation from relational lenses [4] to use $l\nearrow$ for get component and $l\searrow$ for put component of a given lens l. We also assume the following well-behavedness properties of such pairs.

$$l\searrow (s, l\nearrow (s)) = s \text{ for all } s \in S \qquad \text{(GetPut)}$$

$$l\nearrow (l\searrow (s, v)) = v \text{ for all } (s, v) \in S \times V \qquad \text{(PutGet)}$$

Having both properties at the same time guarantees that when either source or view is updated, get or put, respectively, is sufficient to restore consistency between source and view, i.e., round-tripping transformation is not necessary. We use this property to design the workflow of synchronizations.

Cospan composition [13] (a.k.a. Co-targetial composition [6]) of bidirectional transformations are useful for sharing data among data sources having their own private data. For example, lenses on trees [8] use such composition to synchronize bookmarks of different web browsers having their own properties. Dejima network also utilizes this composition for synchronizing base tables owned by peers. In this paper, we use the following figure to

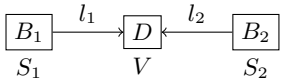
双方向変換の二部ネットワークにおけるビューに基くデータ共有のための適応手法

Soichiro Hidaka, 法政大学情報科学部, Faculty of Computer and Information Sciences, Hosei University.

Hiroyuki Kato, 国立情報学研究所, National Institute of Informatics.

Masato Takeichi, 東京大学, University of Tokyo.

graphically show such compositions.

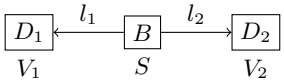


If peer 1 shares its data stored in its base table B_1 of type S_1 in its database with its counterpart peer 2 through a view D of type V, then peer 1 can control the shared information by discarding parts of the information that are not to be disclosed during the forward transformation of lens l_1. Peer 1 can accept updates on shared view D through the backward transformation of l_1. Moreover, peer 2 can also share parts of their own database B_2 of type S_2 using the same view D in a similar manner. The stable situation (i.e., synchronization is already established) in such sharing can be summarized as follows.

$$D = l_i\nearrow(B_i),\ B_i = l_i\searrow(B_i, D)\ i \in \{1, 2\}$$

When peer 1 updates their source database B_1 to B_1', it generates the updated view D' by $l_1\nearrow(B_1')$, and peer 2 accepts the update by generating updated database using $l_2\searrow(B_2, D')$ [11].

Each peer in Dejima network forms a span composition [12] (a.k.a. co-sourcial composition [6]) of bidirectional transformations as follows, by connecting its base table with other peers through their views.

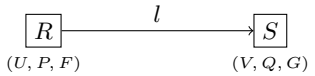


When some other peer that is connected with the view D_1 of type V_1 updates the view, the peer owning base table B of type S accepts the update by backward transformation $l_1\searrow(D_1, B)$, and propagate this change to other set of peers connected with the view D_2 by forward transformation $l_2\nearrow(B')$ where B' is the updated base table.

We found that the synchronization by such compositions of bidirectional transformations, organizing bipartite graphs of bidirectional transformations where base tables form the first set of nodes and the views (Dejimas) form the second set of nodes, while each bidirectional transformation connects one of the first set as a source database and another one of the second set as a view, is quite versatile [3]. However, the network is relatively static in the sense that the bipartite graphs cannot be extended and shrunk easily. Therefore, we propose how to cope with such reorganizations in the bipartite bidirectional transformation network. We can essentially keep the topology of the bipartite graph and can encode various reorganizations, including decomposition and merger of peers, participation of new peers and temporal disconnection of peers, reconciliation of sharing conditions among peers, using additional combinations of bidirectional transformations.

We borrow the type system and dynamic semantics from relational lenses [4] in that we assume that source and target have types associated with constraints as boolean predicates and sets of functional dependencies, and that the type of the view can be statically derived from the type of the source and the transformation.

For the moment, we focus on the type of individual relation to simplify the argument[†1]. Then the type called *sort* is represented by the triple of the set of attributes in the relation, the constraint satisfied by every tuple in the relation, and the set of functional dependencies that are satisfied by the relation.



The above figure shows a lens l of type $(U, P, F) \to (V, Q, G)$ transforms source relation R of type (U, P, F) and view relation S of type (V, Q, G) bidirectionally. Note that we use unidirectional arrows from source to target instead of

†1 Bohannon et al.'s type supports multiple relations in a database. We use that formulation in Section 7.

bidirectional arrows, to clarify the asymmetric nature of l. We may omit the set of attributes and functional dependencies part when these parts are not important or apparent from the context.

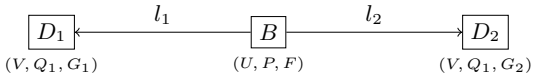
The fundamental idea of the evolution is that we make the nodes and edges evolve, while maintaining the structure of bipartite graph. To achieve this, we classify the evolution (and degeneration) into those of source databases, views and edges, and decompose the evolution (and degeneration) process into the preparatory phase to replace component by their equivalent, and the adaptation phase to conduct essential adaptation.

The rest of the paper is organized as follows. Sections 2 and 3 describe our evolutionary framework for nodes and edges, respectively, for two-party cases. Only selection is involved in the evolved part, because we first assume that adjacent peers already agree on the shared set of attributes. Then we describe their application to reorganization of bipartite bidirectional transformation network in Section 4. Section 5 generalizes our evolutionary framework to multi-party cases. Section 6 extends our framework to cope with differences in the set of attributes. Projection and join are involved in the evolved part for adaptation and initialization. Section 7 describes, given relational operators and their composition, how to reflect changes of constraints in both direction. This is part of our adaptation framework. Section 8 shows a concrete scenario of gracefully extending data sharing that utilizes node evolution. We mention related work in Section 9 and conclude our paper along with our future work in Section 10.

2 Node Evolution

2.1 Source Database Evolution

First is the *preparatory* phase. In case of *source database node evolution*, we can imagine an application of a division (split) of an organization. Suppose, before evolution, that the source database formed a span of two bidirectional transformations l_1 and l_2.

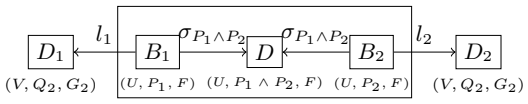


We divide the original source database B into two databases B_1 and B_2 of type (U, P_1, F) and (U, P_2, F), respectively, satisfying $B = B_1 \cup B_2$ and $P = P_1 \vee P_2$.

Then we introduce a cospan of new bidirectional transformations

`select from` B_i `where` $P_1 \wedge P_2$ `as` $D \quad (i \in \{1, 2\})$

centered around the new view D.

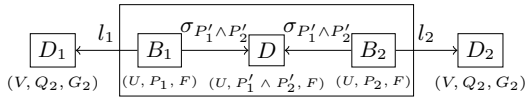


The view D is initialized by the intersection of the divided source databases, i.e., $D = B_1 \cap B_2$. In this paper, we write $R \xrightarrow{\sigma_P} S$ instead of `select from` R `where` P `as` S, and may omit the source and view relations if they are apparent.

Note that, among B_1 and B_2, only the tuples that satisfy P_1 and P_2 are shared via D. Also note that both ends of the introduced cospan keep the interface to the original views D_1 and D_2 unchanged, and that the cospan is interchangeable with the original base table: to degenerate, we remove the new bidirectional transformations in the cospan and restore B as $B_1 \cup B_2$. The special case of this phase is that everything is shared among B_1 and B_2, i.e, $B_1 = B_2 = D = B$ and $P_1 = P_2 = P$.

The *adaptation phase* is concerned with the essential evolution. The "internal" two base tables B_1 and B_2 may change the condition of sharing data with each other. To do so, the conditions

of the bidirectional transformations in the cospan could strengthen or weaken the constraints P_1 and P_2. Note that the bidirectional transformations l_1 and l_2 remain unchanged.



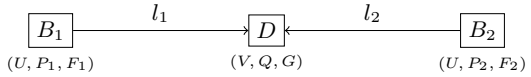
To adapt the contents of the table D to the new predicate $P_1' \wedge P_2'$, D is reinitialized with $\sigma_{P_1' \wedge P_2'}{}^{\nearrow}(B_1) \cup \sigma_{P_1' \wedge P_2'}{}^{\nearrow}(B_2)$. After that, B_1 and B_2 import the change of D by backward transformations of $\sigma_{P_1' \wedge P_2'}$. This is a general workflow of adjusting the transformation change in the cospan composition.

We could further reduce the set of shared columns, i.e., U to $U' \subsetneq U$. We discuss such more general cases in section 6.

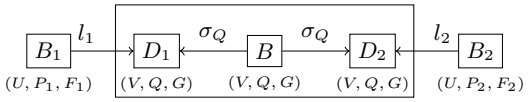
The degeneration to the original base table, that is the cancellation of the evolution, is always possible during this phase, by merging B_1 and B_2 by the union operation. The degenerated node satisfies $P_1 \vee P_2$.

2.2 View Node Evolution

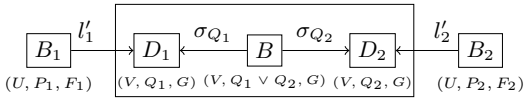
Suppose a view node D is centered-around by a cospan of two bidirectional transformations l_1 and l_2 as follows.



In case of *view node evolution*, the idea is similar to that of source node evolution, but the objective of this evolution is different, in that view evolution can cope with the change of the conditions of sharing via view D. That is, ranges of l_1 and l_2 do not have to agree after evolution.



In the preparatory first phase, we replace the view D by a span of two identical bidirectional transformations σ_Q centered around the new source B. The contents of the two new views D_1, D_2 and the new source B are initialized by the same contents as D. Then in the adaptation phase, when B_1 changes the sharing condition Q to Q_1, then the cospan centered around D_1 correspondingly changes the condition from Q to Q_1. Similarly, another cospan can follow the sharing condition change from Q to Q_2.



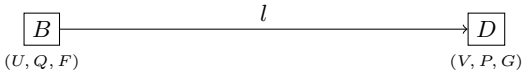
To adapt the contents of the table B to the new predicate, assuming only one side changes at the same time, in this case, say Q_1 side, then B is updated with $\sigma_{Q_1}{}_{\searrow}(B, D_1)$. This update is reflected to D_2 by $\sigma_Q{}^{\nearrow}(B)$. B_2 will then import the update of D_2 by the backward transformation of l_2. The other case is symmetric. This is a general workflow of adjusting the transformation change in a span composition.

After such evolution, tuples that has been sent from B_1 via D_1 make their way to B_2 via D_2 if it satisfies $Q_1 \wedge Q_2$ and the same in the other direction. Tuples that satisfy Q_1 but not Q_2 stays in B. Similarly, tuples that satisfy Q_2 enters the span from B_2 but stays in B if it does not satisfy Q_1.

When Q_1 becomes equal to Q_2 again, the span can be degenerated to the original view D.

3 Edge Evolution

Suppose a base table B and a view D is connected by a bidirectional transformation l as follows.



The purpose of *edge evolution/degeneration* is to cope with the following two scenarios, based on which side of l is fixed while the condition of the other side can change.

One is to keep the constraint of the view D unchanged, while the owner peer of the base table B changes the range of the bidirectional transformation l from P to P'.

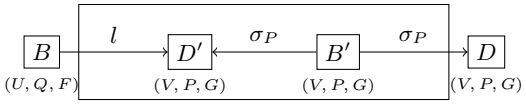
Another scenario is to keep the condition of the tuples to be exchanged by the owner peer of the base table B, but the condition of the view D changes from P to P'.

The first scenario allows a peer to change its sharing condition without forcing the rest of the peers that uses the same view as the first peer to change their bidirectional transformations.

The second scenario allows the view D to change its condition while keeping the range of the bidirectional transformations centered around the view unchanged. This is useful when changing the agreement that was originally made around the view D, without affecting the owner peer of the base table B, because the peer keeps the bidirectional transformation l unchanged, possibly because the peer does not want to change l.

We describe these scenarios in Subsections 3. 2 and 3. 3, after we introduce their common preparatory phase.

3. 1 Preparatory Phase



In the preparatory first phase, we replace the bidirectional transformation l by the combination of l from the original base table B to the new view D' which is initialized by the copy of D, and the span of σ_P between D' and D, centered around newly introduced base table B' which is initialized by the copy of D.

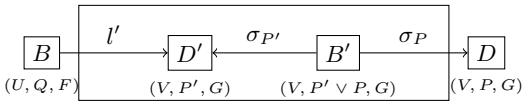
This preparatory phase can be reverted to restore the original connection by directly connecting by l the original base table B and the original view D. This is because the role of the newly synthesized edge l' is equivalent to l. The left-to-right workflow of the span transformation is as follows.

$$B' := \sigma_P \searrow (B', D')$$
$$D := \sigma_P \nearrow (B')$$

The right-to-left workflow is symmetric. Since tuples in D' always satisfy the predicate P, every tuple in D' appears in B' and every tuple in B' appears in D. Therefore, the span is equivalent to identity transformation in both direction. So, the composition of this span with l is equivalent to l.

3. 2 View Condition Preservation

This is achieved in the adaptation phase by changing the bidirectional transformation l while keeping the bidirectional transformation from B' to D unchanged. Suppose the view condition of l is changed from P to P'. This change is absorbed by changing σ_P from B' to D' to $\sigma_{P'}$.



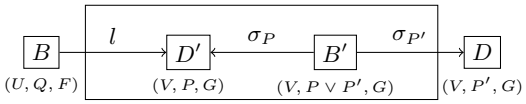
With this evolution, tuples that satisfy the new condition P' make their way from D' to B', but do not reach D if they do not satisfy the original condition P. Similarly, tuples that satisfy the original condition P make their way from D to B' by the backward transformation, but do not reach B if they do not satisfy the new condition P'. In this way, the tuples that satisfy the condition P' but does not satisfy P, or those satisfy the condition P but does not satisfy P' stay in the new base table

B'.

This scenario can be considered as an application of view node evolution if we view the edge evolution as an evolution of view node D that keeps the condition of the new view of the other side of the original base table B.

3.3 Source Condition Preservation

Another scenario is to keep the condition of the tuples to be exchanged by the owner peer of the original base table B, but the condition of the original view changes from P to P'. The condition of the tuples that are accumulated in B' is the same as the first scenario. This is achieved by changing the bidirectional transformation between B' and D from σ_P to $\sigma_{P'}$ in the adaptation phase.



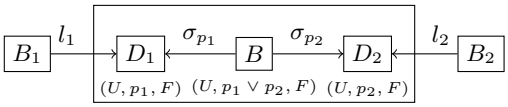
4 Connectivity Applications

We can apply the above mentioned evolution in various case of reorganization of bipartite bidirectional transformation network as follows.

4.1 Inter-peer Connection Introduction

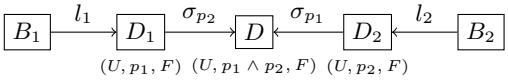
Suppose two peers would like to share some data but do not agree on the condition on the common view. More concretely, peer 1 has a view D_1 with condition p_1, and peer 2 has a view D_2 with condition p_2 for sharing, but p_1 and p_2 are different.

We only consider differences in condition part, i.e., we assume $D_1 :: (U_1, p_1, F_1)$ and $D_2 :: (U_2, p_2, F_2)$ and $U_1 = U_2 = U \wedge F_1 = F_2 = F$. See section 6 for how to cope with $U_1 \neq U_2$. Then we can cope with this situation in one of the following two ways. One is to form a span of bidirectional transformations σ_{p_1} and σ_{p_2} centered around the new base table B that is initialized by $D_1 \cup D_2$.

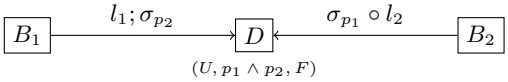


Note that the ranges of the backward transformations of the two bidirectional transformations should agree. We discuss the definition of such agreement in section 7.6.

Another way is to insert a cospan of bidirectional transformations σ_{p_1} and σ_{p_2} centered around the new view D that is initialized by $D_1 \cap D_2$.



The first solution can be seen as an application of view node evolution in the sense that evolved view is inserted between the two peers that originally disagree with their views. The second solution is not particularly associated with evolutionary framework, but still maintains bipartite network structure if we remove the intermediate result of the sequential composition of two bidirectional transformations on each side of the central view.

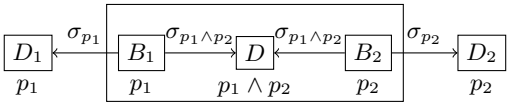


4.2 Inter-view Connection Introduction

Suppose each of two groups of peers are connected with views D_1 and D_2, and that we would like to connect the two views, so that the two groups can share information, but the two views have sharing conditions p_1 and p_2 that are different. Here we omit the set of attributes and the set of functional dependencies.

Then we can insert an evolved base table between

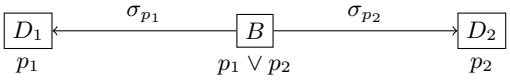
the two views.



The new view D at the center of evolved node is initialized by $D_1 \cap D_2$. The new base tables B_1 and B_2 in the evolved node are initialized by the content of D_1 and D_2, respectively, and these are connected by a cospan of $\sigma_{p_1 \wedge p_2}$ as in the above figure.

Then B_1 and D_1, B_2 and D_2 are respectively connected by σ_{p_1} and σ_{p_2} as in the above figure.

More direct solution would be the following, where the new base table B is initialized by the content of $D_1 \cup D_2$. This corresponds to the configuration after degeneration of the previous configuration.

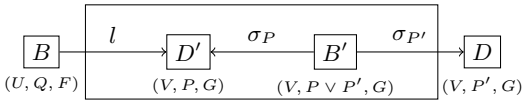


4.3 Database-view Connection Introduction

Suppose there are a pair of base table B with condition Q and a view D with condition P' that are not connected, and would like to be connected.

Applications are an admission of a peer owning the base table B to an existing alliance centered around the view D, or temporal instability of the network that may separate B and D. We can model this situation by an edge evolution/degeneration as follows.

For the first application, we introduce the evolved edge in source condition preservation scenario described in Section 3.3, to cope with possible difference in the view condition on the base table side (P) and that of the view side (P').



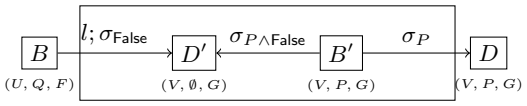
The tables are initialized by the following workflow. Note that, before connection introduction, B and D may not be consistent in general, so the workflow assumes such case.

1. $D' := l\nearrow(B)$
2. $B' := \sigma_P\searrow(D', \emptyset)$
3. $B' := \sigma_{P'}\searrow(B', D)$
4. $D' := \sigma_P\nearrow(B')$

In step 1, original content of B is exported to D'. Step 2 sends the content to B'. Step 3 merges the previous content with the original content of D. Step 4 merges the original content of D into D'.

After these steps, $B := \sigma_P\searrow(B, D')$ imports the contents in D to B, and $D := \sigma_{P'}\nearrow(B')$ imports the contents in B to D.

For the second application, we use a slightly modified edge right after preparatory phase in Section 3.1.



Note that the existence of the boolean expression False effectively keep the original database and the view disconnected (and thus the content of D' is empty), while tuples that satisfies the condition P are accumulated in the table B'. Then we "switch on" the connection by replacing the boolean False to True, and recompute the contents of the center of the cospan and the span, as well as the view D as follows.

1. $D' := l\nearrow(B) \cup \sigma_P\nearrow(B')$
2. $B := l\searrow(D', B)$
3. $B' := \sigma_P\searrow(D', B')$
4. $D := \sigma_P\nearrow(B')$

In step 1, updates on the view and source while dis-

connected are merged in D. In step 2, the updates on the view are reflected to the source. In steps 3 and 4, the updates on the source are reflected to the view.

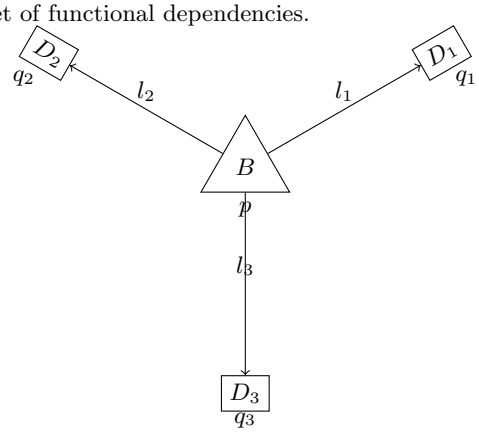
5 Generalization to Multiary Cases

So far, we have discussed the binary cases where only two views or base tables are involved. The generalization to more than two views or base tables are discussed in this section.

Suppose we have three bidirectional transformations

$$l_1 :: (U, p, F) \to (V, q_1, G)$$
$$l_2 :: (U, p, F) \to (V, q_2, G)$$
$$l_3 :: (U, p, F) \to (V, q_3, G)$$

centered around a source base table B. In the following figures, we omit the set of attributes and the set of functional dependencies.



Then the preparatory phase of node evolution introduces a three base tables B_1, B_2 and B_3 that satisfies p_1, p_2 and p_3, respectively, satisfying $B = B_1 \cup B_2 \cup B_3$ and $p = p_1 \vee p_2 \vee p_3$. Ternary multicospan with selection transformation under common predicate $p' = p_1 \wedge p_2 \wedge p_3$ is introduced for a common view D with condition p'.

During the adaptation phase, the selection conditions $p_i, i \in \{1, 2, 3\}$ can be changed to $p'_i, i \in \{1, 2, 3\}$ without changing the original bidirectional transformations l_i, provided that $p' = \bigwedge_i p'_i$ is maintained (Figure 1). Tuples in B_i (already satisfies p_i) flows into B_j if they satisfy the condition $p' = \bigwedge_i p'_i$.

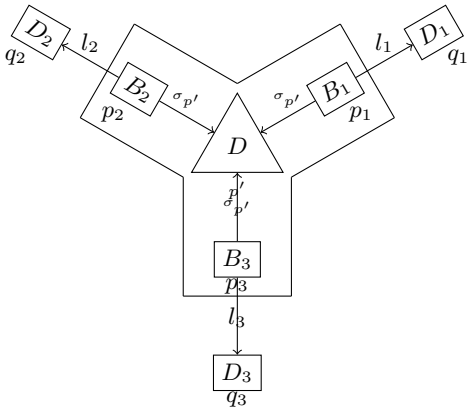
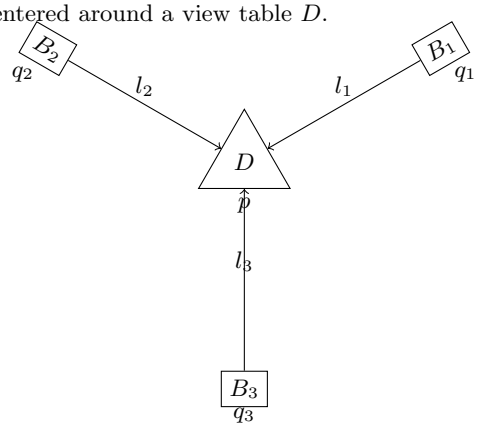


図 1 Evolved multispan node ($p' = p_1 \wedge p_2 \wedge p_3$)

Regarding to the view node evolution, suppose we have three bidirectional transformations

$$l_1 :: (U, q_1, F) \to (V, p, G)$$
$$l_2 :: (U, q_2, F) \to (V, p, G)$$
$$l_3 :: (U, q_3, F) \to (V, p, G)$$

centered around a view table D.



Then in the preparation phase, we introduce a multispan of selection bidirectional transformations $\sigma_{p_i}, i \in \{1, 2, 3\}$ from their common source table B, initialized by the content of D, and p_i are identically initialized to p, while $D_i, i \in \{1, 2, 3\}$ are initialized by the content of D (Figure 2). Then during the adaptation phase, when l_i changes its range to p'_i, the counterpart selection changes its predicate to p'_i. Other peers do not have to change

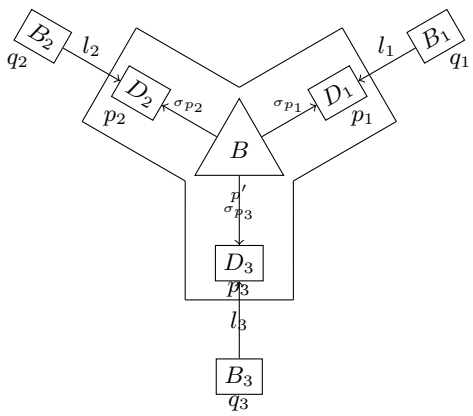


図 2　Evolved multicospan node

$(p' = p_1 \vee p_2 \vee p_3)$

their ranges.

Tuples in D_i flows to D_j through the multispan if they satisfy condition $p_i \wedge p_j$.

When the ranges of $l_i, i \in \{1, 2, 3\}$ agree again, then the evolved node (Y-shape in the figure) is shrunk to one view with the agreed condition p''.

The two evolution scenario can be applied in a row. If we apply view node evolution followed by source node evolution, we have a new view at the center.

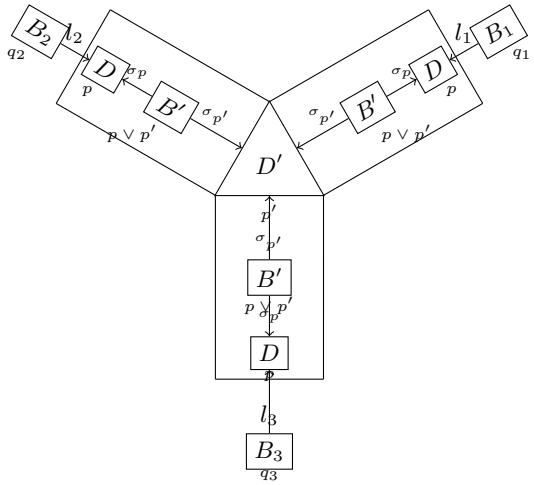


図 3　Edge evolution to cope with alliance agreement change

This configuration can be seen as edge evolution around a view in a multicospan for the view condition change.

6　General Reconciliation of Sorts

In the previous sections, our evolutionary framework considers reconciliation of different constraints as predicates, but did not cope with differences in the set of attributes. For example, in section 2. 1, we only consider two base tables, introduced in source node evolution, that have identical set of attributes. Similarly, in Section 4. 1 we only consider two views that has identical set of attributes.

In this section, we describe how to reconcile sorts that have different constraints as well as different set of attributes. In such situation, common set of attributes are used for data sharing.

To simplify the argument, we assume the sets of functional dependencies are equal at the shared attributes, i.e., we consider two sorts (U_1, P_1, F_1) and (U_2, P_2, F_2) where $U_1 \neq U_2$, $P_1 \neq P_2$, and $F_1' \equiv_U F_2'$ where $U = U_1 \cap U_2$, $F_1' \subseteq F_1$ and $F_2' \subseteq F_2$.

To manipulate set of attributes, bidirectional projection transformation is used. In this paper, we write

$$R \xrightarrow{\pi_{U-A}^{X \to A, a}} S$$

$(U, P[U-A] \bowtie P[A], F \cup \{X \to A\})$　$(U-A, P[U-A], F)$

instead of

$$\texttt{drop}\ A\ \texttt{determined by}\ (X, a)\ \texttt{from}\ R\ \texttt{as}\ S$$

in reference [4], for $A \in U$, $X \in U \setminus A$ and $a \in P[A]$, where source constraint P can be decomposed into conjunction of $P[U-A]$ and $P[A]$. $P[A]$ only refers to attribute(s) in A. We may omit the source and view relations if they are apparent. We revisit this bidirectional transformation with more detailed explanation in Section 7, but intuitively, it removes the column A in the forward direction, and restore the column in the backward direction. On restoration, the values for the column A is restored using

the functional dependency $X \to A$ in the source instance, or the default value a if the value of column X is not found in the source instance. We may further omit the functional dependency $X \to A$ and the default value a. Note that π notation specifies the remaining attributes, while `drop`... notation specifies dropped attributes. Since `drop`... drops only one attribute at a time, projection

$$\boxed{R} \xrightarrow{\pi_{U-V}} \boxed{S}$$
$$(U, P, F) \qquad (U - V, P[U - V], F')$$

involving multiple attributes in set V are represented by the following composition.

$$\underset{A_i \in V}{;} \texttt{drop}\ A_i\ \texttt{determined by}\ (X_i, a_i)\ \texttt{from}\ \_,$$
$$F = F' \cup \bigcup_{A_i \in V} \{X_i \to A_i\}$$

6.1 Cospan Introduction

In the connection among two sources B_1 and B_2 of types (U_1, P_1, F_1) and (U_2, P_2, F_2), respectively, we use the intersection $U = U_1 \cap U_2$ as the common attributes for data sharing.

$$\underset{(U_1, P_1, F_1)}{\boxed{B_1}} \qquad\qquad \underset{(U_2, P_2, F_2)}{\boxed{B_2}}$$

First we introduce projection views to produce views D_1 and D_2 with the intersection attributes U.

$$\underset{(U_1, P_1, F_1)}{\boxed{B_1}} \xrightarrow{\pi_U} \underset{(U, P_1[U], F_1')}{\boxed{D_1}} \qquad \underset{(U, p_2[U], F_2')}{\boxed{D_2}} \xleftarrow{\pi_U} \underset{(U_2, P_2, F_2)}{\boxed{B_2}}$$

Once the views with common set of attributes are set up, remaining process is the same.

$$\underset{(U_1, P_1, F_1)}{\boxed{B_1}} \xrightarrow{\pi_U} \underset{(U, P_1[U], F_1')}{\boxed{D_1}} \xrightarrow{\sigma_{P_2[U]}} \underset{(U, P, F)}{\boxed{D}} \xleftarrow{\sigma_{P_1[U]}} \underset{(U, p_2[U], F_2')}{\boxed{D_2}} \xleftarrow{\pi_U} \underset{(U_2, P_2, F_2)}{\boxed{B_2}}$$

$$P = P_1[U] \cap P_2[U]$$
$$F = F_1' \equiv_U F_2'$$

Note that we require the functional dependencies at the center of the cospan agree.

Given the following compositions

$$l_1 = \pi_U; \sigma_{P_2[U]}$$
$$l_2 = \pi_U; \sigma_{P_1[U]},$$

the initialization of D is achieved by the following workflow.

1. $D := l_1 \nearrow (B_1) \cup l_2 (\nearrow B_2)$
2. $B_1 := l_1 \searrow (B_1, D)$
3. $B_2 := l_2 \searrow (B_2, D)$

Note that the choice of selection after projection instead of projection after selection is intended. In terms of forward transformation, they are interchangeable, because selection refers only to the attributes that are kept by the projection. However, in the backward direction, projection after selection would utilize the default value specified by the projection transformation more than the other choice. Suppose a tuple is inserted in the final view. In the former choice, based on the semantics of Bohannon et al. [4], the backward transformation of projection may insert a default value for the dropped column, if the other tuples do not provide functional dependency that would restore the dropped column. This is more likely than in the other choice, if the selection transformation discard such tuples based on the selection condition. We prefer our choice that more maintain the original functional dependency.

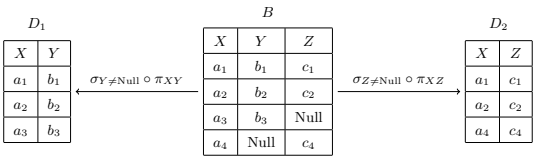
6.2 Span Introduction

In the connection among two views D_1 and D_2 of types indicated in the following figure, we use the union $U = U_1 \cup U_2$ as the set of attributes in the center of the span for data sharing.

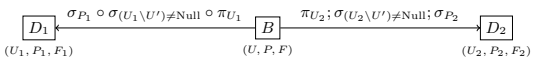
$$\underset{(U_1, P_1, F_1)}{\boxed{D_1}} \qquad\qquad \underset{(U_2, P_2, F_2)}{\boxed{D_2}}$$

If we have the multi-valued dependency $U \twoheadrightarrow (U_1 \backslash U_2) | (U_2 \backslash U_1)$, the central base table can be initialized by $D_1 \bowtie D_2$ (natural join of D_1 and D_2) so that $\pi_{U_1}(D_1 \bowtie D_2) = D_1$ and $\pi_{U_2}(D_1 \bowtie D_2) = D_2$ holds. Otherwise, $D_1 \rtimes\!\ltimes D_2$ (full outer join) can be used. The following figure shows an example span composition using full outer join with predicates using Null values to restore D_1 and D_2 from the

result of full outer join.



Combined with such attributes reconciliation with condition reconciliation, the general span introduction is summarized in the following figure.

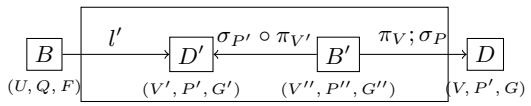


$$U' = U_1 \cap U_2$$
$$P = P_1 \cup P_2$$
$$F = F_1 \cup F_2$$

We assume that F_1 and F_2 do not conflict in F.

6.3 Edge Evolution by Span Introduction

We can extend the view condition preservation scenario of edge evolution in section 3.3 to cope with changes in attribute sets on the source side.



$$V'' = V' \cup V$$
$$P'' = P' \cup P$$
$$G'' = G' \cup G$$

The span around table B' absorbs the attribute sets difference by additionally using projections to keep synchronization using set V'' which is the union of the original attribute set V and set of attributes V' after the change. We assume that G' and G do not conflict in G''.

7 Type and Static BX

So far, we have focused on the cases where constraints on the original base tables before evolution are fixed. However, in general, these constraints may change, for example, when the owner peer decided to reduce or extend the data they want to store. Accordingly, the constraints on the views may change. In the opposite direction, when the constraints on the view changes, the constraints on the source should be changed accordingly. This leads us to consider bidirectional transformations of constraints, which is the focus of this section. In contrast to the transformation of data, we call the constraint transformation *static*. The propagation is based on the type system of Bohannon et al. [4]. In the rest of this section, we recap their type system and introduce the static bidirectional transformations on relations based on Bohannon et al.'s type system.

Bohannon et al.'s bidirectional transformation v between source database schema Σ and view database schema Δ is denoted by $v \in \Sigma \Leftrightarrow \Delta$ where a database schema represent a set of relation names, ranged over $R, S, \ldots$. Database instances, ranged over by $I, J, \ldots$, represents maps from relation names to relations which are sets of tuples, ranged over $m, n, \ldots$, which denote mappings from attribute names in U to attribute values.

As we have already explained, $\text{sort}(R) = (U, P, F)$ denotes a triple of set of attributes U, constraint P and the set of functional dependencies F that are satisfied by the tuples in relation R. Precisely speaking, P in the type system represents the set of all possible tuples that satisfy the constraints. So, the interpretation of P is that $m \in P \Leftrightarrow (\llbracket P \rrbracket_m = \mathsf{True})$. For example, $\llbracket A = \mathrm{a} \rrbracket_{\{A \mapsto \mathrm{a}, B \mapsto \mathrm{b}\}} = \mathsf{True}$.

Function application $\mathit{outputs}(F)$ denotes the set of fields "that are actually constrained by some other fields in the relation" [4]. The proposition "F is in tree form" holds when, intuitively, the topology formed by the graph of disjoint set of attributes in F as nodes and functional dependencies in F between them as edges form a directed acyclic graph. Please refer to reference [4] for their precise definitions.

In the followings, we introduce the general form

of static bidirectional transformations on relations and semantics for relational selection, projection and join operators.

In static bidirectional transformation, forward transformation $\mathcal{F} : \mathit{Expr} \to \mathit{Sort} \to \mathit{Sort}$ computes for relational expression e the sort of view from input (source) sort, whereas the backward transformation $\mathcal{B} : \mathit{Expr} \to (\mathit{Sort} \times \mathit{Sort}) \to \mathit{Sort}$ computes for relational expression e the updated source sort from original source sort and updated view sort. We require these transformations to satisfy the following two properties.

$$\mathcal{F}[\![e]\!]s_{\mathrm{S}} = s_{\mathrm{V}} \Rightarrow \mathcal{B}[\![e]\!](s_{\mathrm{S}}, s_{\mathrm{V}}) = s_{\mathrm{S}} \quad \text{(S-GetPut)}$$

$$s'_{\mathrm{S}} = \mathcal{B}[\![e]\!](s_{\mathrm{S}}, s_{\mathrm{V}}) \Rightarrow \mathcal{F}[\![e]\!]s'_{\mathrm{S}} = s_{\mathrm{V}} \quad \text{(S-PutGet)}$$

Property S-GetPut says, similarly to original GetPut, that source sorts are unchanged after using backward transformation of the view sorts that are not updated. Property S-GetPut says, similarly to original PutGet, that view sorts are unchanged after backward transformation followed by the forward transformation using the updated source sorts.

We also aim at, though not formalized, least change in the backward direction.

Now we explain selection, join and projection transformations. We fix the relational operators, set of attributes and set of functional dependencies in the transformation.

7.1 Static Selection

According to the typing rule T-Select for selection [4], the constraint of the view is the conjunction of that of the source relation and the predicate in the selection transformation.

In the backward direction, the updated view constraints should still imply P (otherwise, tuples in the updated view deviates the constraint P), so, given the new view constraint X', $X' \subseteq P$. So the updated source constraint will be computed by $Q \setminus P \cup X'$.

$$\frac{\begin{array}{c}\mathit{sort}(R) = (U, Q, F) \\ \mathit{sort}(S) = (U, P \cap Q, F) \\ F \text{ is in tree form} \quad Q \text{ ignores } \mathit{outputs}(F)\end{array}}{\begin{array}{c}\texttt{select from } R \texttt{ where } P \texttt{ as } S \in \\ \Sigma \uplus \{R\} \Leftrightarrow \Sigma \uplus \{S\}\end{array}} \quad \text{(T-Select)}$$

$$\mathcal{F}[\![\sigma_P]\!]((U, Q, F)) = (U, X, F), X = P \cap Q$$

$$\mathcal{B}[\![\sigma_P]\!]((U, Q, F), (U, X', F)) = (U, Q \setminus P \cup X', F)$$

$$X' \subseteq P$$

$$F \text{ is in tree form}$$

$$Q, X' \text{ ignores } \mathit{outputs}(F)$$

With respect to the least change principle, the change of the source constraint is within the set representation of P, so there is no extra change that is invisible on the view. Therefore, the backward semantics $\mathcal{B}[\![\sigma_P]\!]$ complies with the principle.

7.2 Static Join

According to T-Join in reference [4], the view constraint of the join of two relations with constraints P on attributes U, and constraints Q on attributes V, is $P \bowtie Q$ which is a constraint that are decomposable into conjunction of the constraints on attributes U (denoted by $(P \bowtie Q)[U]$) and constraints on attributes on V (denoted by $(P \bowtie Q)[V]$). If there is no such predicate on either side, that will be considered True. For example, when $U = \{a, b\}$ and $V = b, c$, $X = a < b \wedge b < c$ is decomposable into $X[U] = a < b$ and $X[V] = b < c$, so $X = (a < b) \bowtie (b < c)$. However $a < c$ is not decomposable because the predicate $a = c$ extends across U and V.

Therefore, in the backward direction, given the updated constraint $P' \bowtie Q'$ that are decomposable into that on U and V, the updated constraints are P' and Q' on U and V, respectively, which satisfies the same constraints with respect to the functional dependencies as P and Q, respectively. Note that, in the backward direction, the constraints of the

original source relations are not used.

$$\frac{\begin{array}{c} sort(R) = (U, P, F) \quad sort(S) = (V, Q, G) \\ sort(T) = (UV, P \bowtie Q, F \cup G) \\ G \models U \cap V \to V \\ F \text{ is in tree form} \quad G \text{ is in tree form} \\ P \text{ ignores } outputs(F) \quad Q \text{ ignores } outputs(G) \end{array}}{\texttt{join\_dl}\ R, S\ \texttt{as}\ T\ \in\ \Sigma \uplus \{R, S\} \Leftrightarrow \Sigma \uplus \{T\}} \text{(T-Join)}$$

$$\mathcal{F}[\![\bowtie]\!]((U, P, F), (V, Q, G)) = (UV, P \bowtie Q, F \cup G)$$
$$\mathcal{B}[\![\bowtie]\!](((U, P, F), (V, Q, G)), (UV, P' \bowtie Q', F \cup G)) = ((U, P', F), (V, Q', G))$$
$$R \bowtie S \stackrel{\text{def}}{=} \texttt{join\_dl}\ R, S$$
$$G \models U \cap V \to V$$
$$F \text{ is in tree form} \quad G \text{ is in tree form}$$
$$P, P' \text{ ignores } outputs(F)$$
$$Q, Q' \text{ ignores } outputs(G)$$

7.3 Static Projection

According to T-Drop in reference [4], projection on the set of columns U in which a column A is removed in the forward direction, given the source constraint P that is decomposable into $P[U - A]$ and $P[A]$, the view constraint is $P[U - A]$. Therefore, in the backward direction, the new view constraint should still refer only to the set $U - A$, and that part is reflected to the source constraint, while A component of the constraint should remain unchanged.

$$\frac{\begin{array}{c} sort(R) = (U, P, F) \\ A \in U \quad F \equiv F' \cup \{X \to A\} \\ sort(S) = (U - A, P[U - A], F') \\ P = P[U - A] \bowtie P[A] \quad \{A = a\} \in P[A] \end{array}}{\texttt{drop}\ A\ \texttt{determined by}\ (X, a)\ \texttt{from}\ R\ \texttt{as}\ S \in \Sigma \uplus \{R\} \Leftrightarrow \Sigma \uplus \{S\}} \text{(T-Drop)}$$

$$\mathcal{F}[\![\pi_{U-A}]\!]((U, P[U - A] \bowtie P[A], F' \cup \{X \to A\})) = (U - A, P[U - A], F')$$
$$\mathcal{B}[\![\pi_{U-A}]\!]((U, P, F), ((U - A, P'[U - A], F'))) = ((U, P'[U - A] \bowtie P[A], F' \cup \{X \to A\}))$$
$$A \in U \quad F \equiv F' \cup \{X \to A\}$$
$$P = P[U - A] \bowtie P[A] \quad \{A = a\} \in P[A]$$
$$P' = P'[U - A] \bowtie P'[A] \quad P'[A] = P[A]$$

The condition $P'[A] = P[A]$ is required for the least change principle. Note that, when we write the updated view sort as $(U - A, P'[U - A], F')$, we assume that $P'[U - A]$ satisfies the set of functional dependencies F'.

7.4 Static Composition

The typing rule of composition T-Compose in reference [4] requires that the view schema of the first bidirectional transformation and the source schema of the second bidirectional transformation are equal.

$$\frac{v \in \Sigma \Leftrightarrow \Sigma' \qquad w \in \Sigma' \Leftrightarrow \Delta}{v; w \in \Sigma \Leftrightarrow \Delta} \text{(T-Compose)}$$

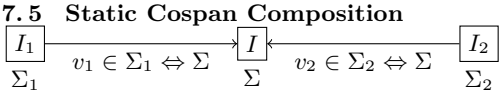
Two schema Σ_1 and Σ_2 being equal means the sorts associated with the set of relations are equivalent.

$$\forall R_1 \in \Sigma_1 . \forall R_2 \in \Sigma_2 . \; R_1 = R_2 \Rightarrow sort_1(R_1) \equiv sort_2(R_2)$$
$$sort_1(R_1) \equiv sort_2(R_2) \stackrel{\text{def}}{=} sort_1(R_1) = (U, P, F) \wedge sort_2(R_2) = (U, Q, G) \wedge F \equiv_U G \wedge \quad P \equiv_U Q$$
$$P \equiv_U Q \stackrel{\text{def}}{=} \forall m : U. [\![P]\!]_m \Leftrightarrow [\![Q]\!]_m$$

To propagate constraint changes along with this rule, the forward transformation of the static composition uses the updated view constraints resulted from the static forward transformation of the first bidirectional transformation as the source constraints of the static forward transformation of the second bidirectional transformation. Similarly, the backward transformation of the static composition creates the updated intermediate constraint as the result of the static backward transformation of the second bidirectional transformation that uses the original intermediate constraint resulted from the static forward transformation of the first bidirectional transformation, and the updated target constraints. The updated intermediate constraint is then used as the updated target constraint of the static backward transformation of the first bidirectional transformation.

$$\mathcal{F}[\![l_1; l_2]\!]s_{\mathrm{S}} = \mathcal{F}[\![l_2]\!](\mathcal{F}[\![l_1]\!]s_{\mathrm{S}})$$
$$\mathcal{B}[\![l_1; l_2]\!](s_{\mathrm{S}}, s_{\mathrm{V}}) = \mathcal{B}[\![l_1]\!](s_{\mathrm{S}}, \mathcal{B}[\![l_2]\!](\mathcal{F}[\![l_1]\!]s_{\mathrm{S}}, s_{\mathrm{V}}))$$

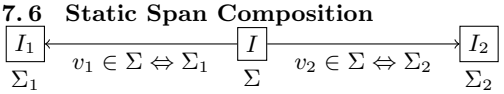
7.5 Static Cospan Composition



The typing rule for the cospan composition of bidirectional transformations v_1 and v_2 requires that the view schema of v_1 and v_2 are equal.

Correspondingly, the static forward propagation of constraint is achieved by $\mathcal{F}[\![v_1]\!]$ followed by $\mathcal{B}[\![v_2]\!]$, and the opposite direction is symmetric.

7.6 Static Span Composition



The typing rule for the span composition of bidirectional transformations v_1 and v_2 requires that the source schema of v_1 and v_2 are equal.

Correspondingly, the static forward propagation of constraint is achieved by $\mathcal{B}[\![v_1]\!]$ followed by $\mathcal{F}[\![v_2]\!]$, and the opposite direction is symmetric.

8 Concrete Example

Figure 4 shows a concrete application of node evolution and degeneration ((a) to (f)). In this hypothetical scenario, two instructors Hidaka and Kato who belong to the same faculty are going to teach a common subject for each department which are Computer Science (CS for short) and Information Systems Architecture Science (IS for short), respectively. Every student in the faculty belongs to either CS or IS. We write *DEPARTMENT* to mean a student belong to department *DEPARTMENT*. The instructors also lead their own laboratories (HL and KL). This situation is illustrated in Figure 4 (a). The formula $CS \vee IS$ below the box of B shows that every student belongs to either CS or IS. To prepare classes for each department, the base table storing student data of the faculty is disjointly split by source node evolution (b). Notice the new constraints on each base table being *CS* and *IS*, sharing condition by the cospan BX is $CS \wedge IS$. Since the two departments are disjoint, initial view is empty. Then the two instructors decided to share information on students belonging to each laboratory. Note that such students may belong to either of the department. Therefore, each instructor prepare to change the sharing condition by the preparation phase of view node evolution (c). Nothing is shared yet at this stage. Then the instructors extend the (provision) range by changing the selection condition of BX by disjunction of $(CS \wedge KL)$ for Hidaka, and $(IS \wedge HL)$ for Kato (d). Note that the difference of sharing conditions in both sides are absorbed by the span at the center. We run the workflow in the figure to reflect the change, i.e., to actually export data about students that belong to the other laboratory. The order in the workflow is relevant.

After the workflow, the data in bow tie shape appear in the central base table B. Even at this stage, no update is propagated among B_1 and B_2 because no instructor receives the data exported by the other instructor yet. So the instructors extend the receiving range by extending the condition of the central span, followed by static backward transformation of the selection at both ends (e). That extends the condition of each base table: *CS* to $CS \vee HL$ for B_1, and *IS* to $IS \vee KL$ for B_2. The workflow in the figure is conducted to propagate applicable tuples.

Now that all the conditions of the bidirectional transformations are equal, the evolved node is ready to degenerate. After refactoring the formula, the degenerated configuration is shown in Figure 4 (f).

To create this state immediately after (b), we could have initialized each table by the following

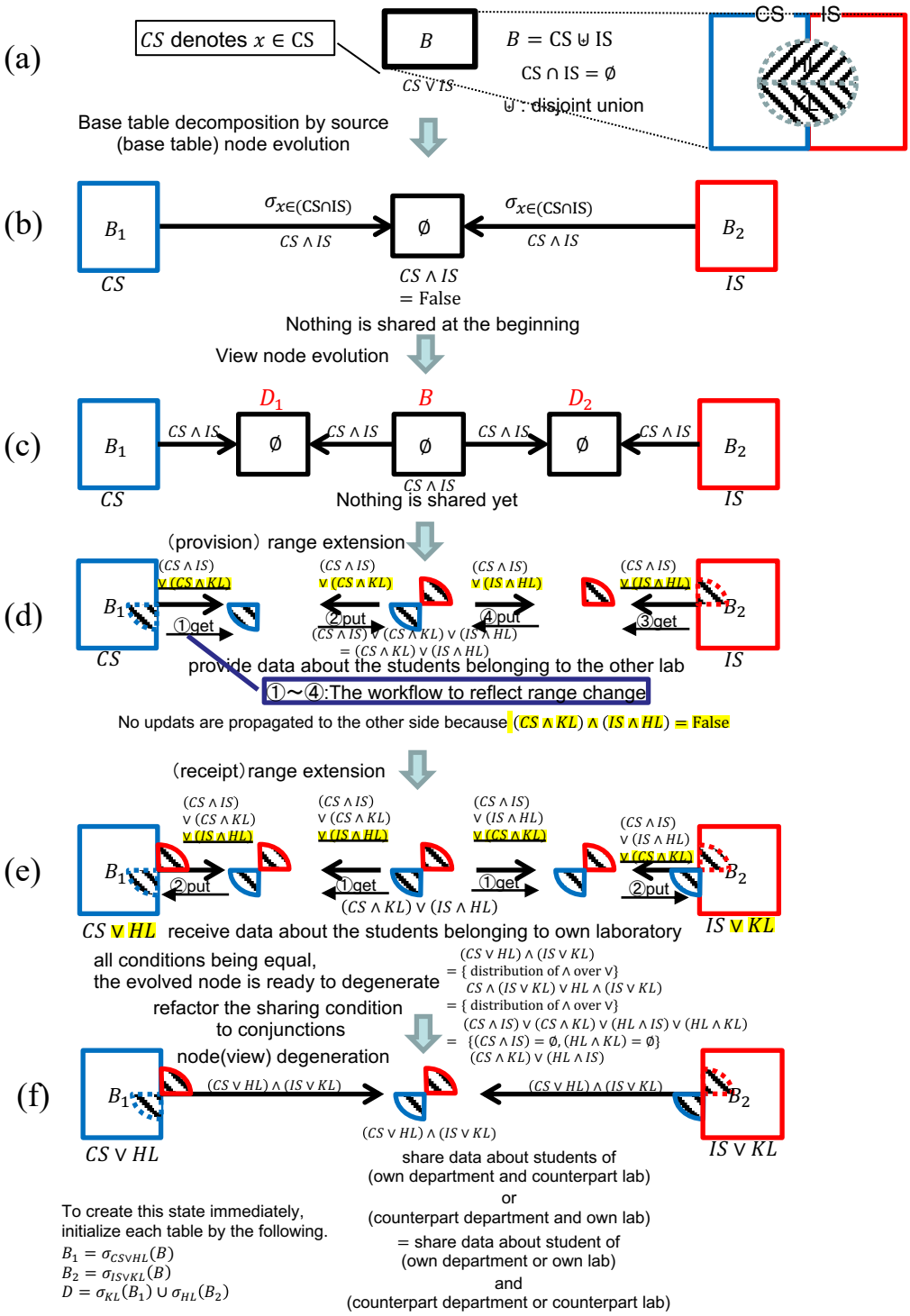


図 4　Node evolution example

workflow.

1. $B_1 := \sigma_{CS \vee HL}(B)$
2. $B_2 := \sigma_{IS \vee KL}(B)$
3. $D := \sigma_{KL}(B_1) \cup \sigma_{HL}(B_2)$

9 Related Work

Asano et al. [3] summarized their BISCUITS project led by Zhenjiang Hu on Software Foundations for Interoperability of Autonomic Distributed Data Based on Bidirectional Transformations. The project is based on Dejima architecture [11], which is a bipartite bidirectional transformation network. Present paper assumes the architecture. However, the architecture itself does not support systematic reconfiguration as we propose in this paper.

In Section 2.3 of the summary paper, they briefly propose to replace a view (Dejima group) in bipartite bidirectional transformation network by a multispan (new dedicated peer) when more expressive power of synchronization was needed. In the present paper, this idea is formalized as a view node evolution in Section 2. Now our framework is more general to cover source node evolution as well as edge evolution. Asano et al. [3] also briefly mention in the same section on how to reconcile ranges of bidirectional transformations on both sides of cospan composition. They propose to use conjunction of each other's constraint, as we followed in Section 4. 1. However, they did not provide more systematic framework for rage reconciliation as we do in this paper.

10 Conclusion and Future Work

In this paper, we proposed an adaptation framework for bipartite network of bidirectional transformations. The idea is based on gracefully inserting (multi)spans and (multi)cospans of bidirectional transformations, gracefully in the sense that node and edge insertions are achieved without changing the topology of the bipartite graph nor existing sharing constraints at the connected nodes. Spans were useful to absorb differences in sharing conditions and set of attributes, while cospans were used in combination with conjunction of constraints and intersection of shared attributes. The constraints satisfied by the tuples that flow through the adapted components were also formalized. The static propagation of constraint changes were proposed as static bidirectional transformation for selection, join, projection and their composition, to complement the evolution. Concrete application scenario was shown to demonstrate effectiveness of our adaptation framework.

There is plenty of room in our framework for improvements. In range reconciliation, we assumed the set of functional dependencies already agreed. It is not even clear to us what is meant by functional dependency reconciliation. In the static bidirectional transformation, column and functional dependency changes were not discussed. In the center of span and cospan composition, conflict may arise for simultaneous propagation of changes. We only consider cases where only one of participating bidirectional transformations updates the central table, while other participants accept changes. Resolving conflict was not considered in this paper. The combination of operational transformation [7] and bidirectional transformation was proposed, for example in reference [9]. However, global consistency is not explicitly considered as Dejima architecture did by two-phase commit and two-phase locking, and BCDS Agent [14] did by first writer wins. We have no implementation of our framework. We would like to provide software modules for evolved nodes and edges. These belong to our future work.

Acknowledgments This work is partially supported by by JSPS Kakenhi 21H03419, 17H06099 (BISCUITS project) and JST CREST JPMJCR21M4. We thank BISCUITS project members for their constructive comments and sugges-

tions.

References

[1] Abou-Saleh, F., Cheney, J., Gibbons, J., McKinna, J., and Stevens, P.: *Introduction to Bidirectional Transformations*, Lecture Notes in Computer Science, Vol. 9715, Springer International Publishing, 2018, pp. 1–28.

[2] Anjorin, A., Diskin, Z., Wang, M., and Xiong, Y.: Bidirectional Transformations, (NII Shonan Meeting 2016-13), *NII Shonan Meet. Rep.*, Vol. 2016(2016).

[3] Asano, Y., Cao, Y., Hidaka, S., Hu, Z., Ishihara, Y., Kato, H., Nakano, K., Onizuka, M., Sasaki, Y., Shimizu, T., Takeichi, M., Xiao, C., and Yoshikawa, M.: Bidirectional Collaborative Frameworks for Decentralized Data Management, *Software Foundations for Data Interoperability*, Fletcher, G., Nakano, K., and Sasaki, Y.(eds.), Cham, Springer International Publishing, 2022, pp. 13–51.

[4] Bohannon, A., Pierce, B. C., and Vaughan, J. A.: Relational Lenses: A Language for Updatable Views, *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, New York, NY, USA, Association for Computing Machinery, 2006, pp. 338–347.

[5] Czarnecki, K., Foster, J. N., Hu, Z., Lämmel, R., Schürr, A., and Terwilliger, J.: Bidirectional Transformations: A Cross-Discipline Perspective, *ICMT*, 2009, pp. 260–283.

[6] Diskin, Z.: Algebraic Models for Bidirectional Model Synchronization, *MODELS*, Czarnecki, K., Ober, I., Bruel, J., Uhl, A., and Völter, M.(eds.), 2008, pp. 21–36.

[7] Ellis, C. A. and Gibbs, S. J.: Concurrency Control in Groupware Systems, *Proc. SIGMOD'89*, 1989, pp. 399–407.

[8] Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., and Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem, *ACM Transactions on Programming Languages and Systems*, Vol. 29, No. 3(2007), pp. 17.

[9] Habu, M. and Hidaka, S.: Conflict Resolution for Data Updates by Multiple Bidirectional Transformations, *Software Foundations for Data Interoperability*, Fletcher, G., Nakano, K., and Sasaki, Y.(eds.), Cham, Springer International Publishing, 2022, pp. 62–75.

[10] Hu, Z., Schürr, A., Stevens, P., and Terwilliger, J. F.: Bidirectional Transformation "bx" (Dagstuhl Seminar 11031), *Dagstuhl Reports*, Vol. 1, No. 1(2011), pp. 42–67.

[11] Ishihara, Y., Kato, H., Nakano, K., Onizuka, M., and Sasaki, Y.: Toward BX-based Architecture for Controlling and Sharing Distributed Data, *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2019, pp. 1–5.

[12] Johnson, M. and Rosebrugh, R. D.: Spans of lenses, *CEUR@EDBT/ICDT*, Candan, K. S., Amer-Yahia, S., Schweikardt, N., Christophides, V., and Leroy, V.(eds.), 2014, pp. 112–118.

[13] Johnson, M. and Rosebrugh, R. D.: Cospans and symmetric lenses, *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, Marr, S. and Sartor, J. B.(eds.), 2018, pp. 21–29.

[14] Takeichi, M.: BCDS Agent: An Architecture for Bidirectional Collaborative Data Sharing, *Computer Software*, Vol. 38, No. 3(2021), pp. 3_41–3_57.