Math. Struct. in Comp. Science (2002), *vol.* 12, *pp.* 349–376. © 2002 Cambridge University Press
DOI: 10.1017/S0960129501003553 Printed in the United Kingdom

Concurrency and loose semantics of open graph transformation systems[†]

REIKO HECKEL[‡], MERCÈ LLABRÉS[§], HARTMUT EHRIG[¶], and FERNANDO OREJAS[‖]

[‡]*University of Paderborn, Germany*
Email: reiko@uni-paderborn.de

[§]*University of the Balearic Islands, Palma de Mallorca, Spain*
Email: merce@ipc4.uib.es

[¶]*Technical University of Berlin, Germany*
Email: ehrig@cs.tu-berlin.de

[‖]*Technical University of Catalonia, Spain*
Email: orejas@lsi.upc.es

Received 20 July 2000; revised 15 February 2001

Graph transitions represent an extension of the DPO approach to graph transformation for the specification of reactive systems. In this paper, we develop the theory of concurrency for graph transitions. In particular, we prove a local Church–Rosser theorem and define a notion of shift-equivalence that allows us to represent both intra-concurrency (within the specified subsystem) and inter-concurrency (between subsystem and environment). Via an implementation of transitions in terms of DPO transformations with context rules, a second, more restrictive notion of equivalence is defined that captures, in addition, the extra-concurrency (between operations of the environment). As a running example and motivation, we show how the concepts of this paper provide a formal model for distributed information systems.

1. Introduction

The typical architecture for distributed information systems, such as electronic banking systems, e-commerce systems, or online reservation systems, is that of a client-server application based on a server that provides, through a database connection, services for the integrated storage and concurrent access of the clients data. Several client applications working concurrently on different views of the overall database may invoke these services by reading and updating the data. Moreover, client applications may interact among each other through the database state by observing the effects of each other's updates, provided that their views are not disjoint.

[†] This paper is a complete and revised version of a lecture given at the workshop on Theory and Applications of Graph Transformations (GRATRA 2000), which was a satellite event of ETAPS 2000 in Berlin, March 2000.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

In general, each application has only incomplete information about the structure and state of the overall database. And although the effects of operations of other applications may be observed, the applications themselves and their operations remain invisible. This loose coupling has the advantage that applications may be added or removed while the system is running without violating the consistency of others.

However, it also means that a client application has to be prepared to work in a largely unknown environment. In fact, all that is typically fixed at development time is the local view of the client, that is, a projection of the overall database schema, and the client's own operations. The effects of the operations of the environment (that is, other clients) on the local view may at most be constrained by requiring explicit permissions for reading and/or updating certain data.

This lack of information raises semantic questions for the behavioural modelling, testing, and verification of client applications: What is the operational interpretation of models of client views? What does a local test of a client application tell us about its behaviour in a larger environment? Which properties are preserved under this embedding? To answer these questions, a new approach to system modelling is required that distinguishes between the *view*, which is specified, and the *environment*, which is not specified, and that acts concurrently with the view and affects its state in an only partly predictable way (Heckel *et al.* 1999).

In this paper, we will contribute to the semantic foundations of view-based specification. Following Heckel *et al.* (2001) and Heckel (1998), we will formalize the idea of under-specified behaviour by the notion of *graph transition*, a generalization of the double-pushout (DPO) approach to graph transformation (Ehrig *et al.* 1973). We will analyze the concurrency within and between client views in relation to the overall system and answer some of the questions raised above.

Employing graph transformation as a semantic model of information systems, we use graphs conforming to a type graph (Corradini *et al.* 1996) as a representation of database states for a given schema, and graph transformation rules to model query and update operations. The PROGRES environment (Schürr *et al.* 1999) and the use of graphs as visual queries (Andries and Engels 1996) provide practical examples of this interpretation. However, classical graph transformation systems (like most rule-based approaches) come equipped with an *implicit frame condition* that ensures that nothing is changed during a transformation beyond what is explicitly specified. Thus, they are not suitable for modelling unspecified effects of the environment because they assume complete control of the system and its behaviour.

Graph transitions have been proposed to provide a looser interpretation of graph transformation rules. A graph transition preserves, deletes and adds at least what is specified by the rule, but it allows additional changes, which are interpreted as effects of the operation of the environment. Technically speaking, graph transitions are based on a double-pullback (DPB) construction, unlike the classical transformations defined by a double-pushout (DPO).

One of the most important services that a database system provides to its clients in a distributed information system is the synchronization of concurrency. Usually, a locking mechanism is employed by the scheduler, which allows concurrent read but ensures

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

exclusive write access to individual data items. The desired semantic property is that each actual schedule is *conflict-equivalent* to a sequential schedule (Bernstein *et al.* 1987). This means that all conflicts between write and read or write requests are decided in a way that would permit a purely sequential execution. In this way, the concurrency improves the performance and availability of the database system, but does not produce additional effects beyond that of a sequential execution.

The striking similarity with the basic concepts of independence and parallelism of graph transformations (see Kreowski (1977), Corradini *et al.* (1996) or Baldan *et al.* (1999) for a survey) tempts us to extend our semantic model of information systems to include, besides database schema and states, update operations and views, also the concurrency aspect. In fact, the database policy of 'concurrent read but exclusive write' captures the idea of (parallel or sequential) independence of transformation steps, and conflict equivalence is conceptually similar to the notion of *shift equivalence*, which captures true concurrency in the DPO approach (Kreowski 1977). Finally, each sequence of parallel DPO steps may be transformed into a shift-equivalent sequential transformation.

Therefore, in this paper we propose a truly concurrent semantics for view-based specification. In particular, we develop the basic notions of independence and shift-equivalence of *faithful* graph transitions, that is, transitions whose bottom span is injective and whose match and comatch satisfy the identification condition of the DPO approach. We shall prove a local Church–Rosser theorem, which ensures the 'correctness' of our notion of independence, and we shall define a corresponding notion of shift-equivalence on transition sequences and study its compatibility with the notion of view. Conceptually, this will allow us to represent both intra-concurrency (within a local view) and inter-concurrency (between the view and its environment). Based on an implementation of transitions by DPO steps with context rules, a second notion of equivalence and independence is defined that takes into account, in addition, the extra-concurrency (between operations of the environment).

After providing in the next section the basic definitions and intuitions about graph transformation for modelling concurrent information systems, in Section 3 we recall the concept of views of graph transformation systems and their loose semantics in terms of graph transitions. In Section 4, the independence of graph transitions is studied, a local Church–Rosser theorem is proved, and a corresponding notion of shift-equivalence is defined. In Section 5, we provide an implementation of faithful graph transitions by DPO transformations, derive a notion of implementation equivalence for transitions, and compare the corresponding notion of independence of transitions with that defined in Section 4. Finally, we give conclusions in Section 6.

2. System modelling with graph transformation

In this section, we introduce and exemplify the basic definitions of the double-pushout (DPO) approach to graph transformation (Ehrig *et al.* 1973) (see Corradini *et al.* (1997) for a survey) and review part of the theory of the approach, in particular, the notions of independence and of shift-equivalence.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

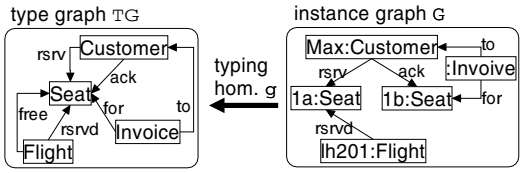


Fig. 1. Type and instance graph.

2.1. *Graphs and typed graphs*

The distinction between schema and instance level was first introduced in database models like the relational or the entity-relationship model and are nowadays fundamental to object-oriented modelling. In the DPO approach to graph transformation, this distinction is formalized by the notions of *type* and *instance graph* (Corradini *et al.* 1996), categorically a comma category over the category of graphs. Thus, we first have to recall the basic definitions of graphs and graph morphisms.

By *graphs* we mean directed unlabelled graphs $G = \langle G_V, G_E, src^G, tar^G \rangle$ with set of vertices G_V, set of edges G_E, and functions $src^G : G_E \to G_V$ and $tar^G : G_E \to G_V$ associating with each edge its source and target vertex. A graph homomorphism $f : G \to H$ is a pair of functions $\langle f_V : G_V \to H_V, f_E : G_E \to H_E \rangle$ preserving source and target, that is, $src^H \circ f_E = f_V \circ src^G$ and $tar^H \circ f_E = f_V \circ tar^G$. The *category of graphs and graph morphisms* with composition and identities defined component-wise on the underlying functions is denoted by **Graph**.

Definition 2.1 (Typed graphs). Given a graph TG, called *type graph*, a *TG-typed (instance) graph* consists of a graph G together with a typing homomorphism $g : G \to TG$ (*cf.* Figure 2 on the left) associating with each vertex and edge x of G its type $g(x) = t$ in TG. In this case, we also write $x : t \in G$. A TG-typed graph morphism between two TG-typed instance graphs $\langle G, g \rangle$ and $\langle H, h \rangle$ is a graph morphism $f : G \to H$ that preserves types, that is, $h \circ f = g$. The category of all instance graphs typed over TG is denoted by $\mathbf{Graph}_{TG}$.

Categorically speaking, $\mathbf{Graph}_{TG}$ is the *comma category* $(\mathbf{Graph} \downarrow TG)$ *of graphs over* TG (see, for example, Mac Lane (1971)). This observation allows the inheritance of many categorical constructions from **Graph** to $\mathbf{Graph}_{TG}$. In particular, since the category of graphs has all limits and colimits, the comma category $\mathbf{Graph}_{TG}$ has all limits and colimits as well, and their constructions coincide in **Graph** and $\mathbf{Graph}_{TG}$ up to the additional typing information.

Example 2.2 (Type and instance graphs). Figure 1 shows the type graph and a corresponding instance graph for the running example of this paper, a flight reservation system. We may think of a type graph as a conceptual database schema, while an instance graph represents a particular state of the database. We use UML notation (Object Management Group 1999) for class and object diagrams, respectively. Thus, v:t denotes a vertex v of type t. The name of the vertex will be omitted if it is of no interest.

The type graph introduces the vertex types Customer, Seat, Flight and Invoice. The

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

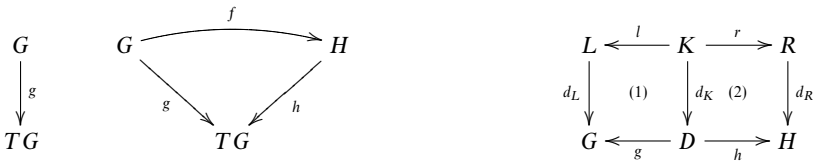


Fig. 2. Typed graph and graph morphism (left) and double-pushout diagram (right).

edge types rsrv, ack from Customer to Seat as well as free, rsrvd from Flight to Seat represent the status of a seat reservation. An Invoice is equipped with two edges pointing to a Customer to whom it is addressed, and to the Seat that is to be paid for.

An instance graph of this type graph is also shown, with customer Max having received an (anonymous) Invoice for the reservation of seat 1b, which he has acknowledged, while a second seat 1a has only been reserved.

2.2. *Typed graph transformations*

The DPO approach to graph transformation was originally developed for vertex- and edge-labelled graphs (Ehrig *et al.* 1973). Here, we present the typed version (Corradini *et al.* 1996).

According to the DPO approach, graph transformation rules, also called graph productions[†], are specified by pairs of injective graph morphisms ($L \stackrel{l}{\longleftarrow} K \stackrel{r}{\longrightarrow} R$), called rule spans. The *left-hand side* L contains the items that must be present for an application of the rule, the *right-hand side* R those that are present afterwards, and the *context graph* K specifies the 'gluing items', that is, the objects that are read during application, but are not consumed.

Definition 2.3 (Rule, graph transformation system). A *rule span typed over* TG, in short TG-typed rule span, $s = (L \stackrel{l}{\longleftarrow} K \stackrel{r}{\longrightarrow} R)$ is a span of injective TG-typed graph morphisms.

A *graph transformation system* $\mathcal{G} = \langle TG, P, \pi \rangle$ consists of a *type graph* TG, a set of rule names P, and a mapping π associating with each rule name p a TG-typed rule span $\pi(p)$. If $p \in P$ is a rule name and $\pi(p) = s$, we say that $p : s$ *is a rule of* $\mathcal{G}$.

The transformation of graphs is defined by a pair of pushout diagrams, a so-called double pushout.

Definition 2.4 (DPO graph transformation). Given a type graph TG, a TG-typed graph transformation rule is a pair $p : s$ consisting of a *rule name* p and a *rule span* $s = (L \stackrel{l}{\longleftarrow} K \stackrel{r}{\longrightarrow} R)$.

A *double-pushout (DPO) diagram* d is a diagram as in Figure 2 on the right, where (1)

[†] In general, we speak of graph productions and graph grammars if we are interested in the generation of graph languages, and of rules and graph transformation system if the focus is on the transformation process itself.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

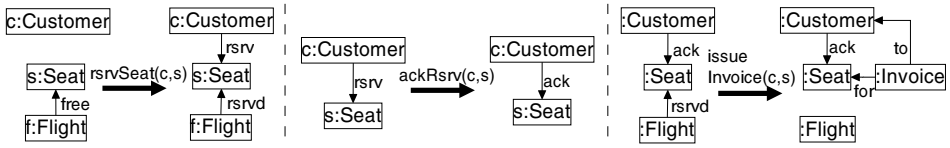


Fig. 3. Typed graph transformation system.

and (2) are pushouts. Given a rule $p : s$ as above, the corresponding *(DPO) transformation step* from G to H is denoted by $G \overset{p/d}{\Longrightarrow} H$, or simply $G \overset{p}{\Longrightarrow} H$ if the diagram d is understood.

A *transformation (sequence)* $\rho = \rho_1 \ldots \rho_n : G \Rightarrow H$ in $\mathscr{G}$ is a sequence of transformation steps $\rho_i = G_i \overset{p_i/d_i}{\Longrightarrow} H_i$ such that $G_1 = G, H_n = H$ and consecutive steps are composable, that is, $G_{i+1} = H_i$ for all $1 \leqslant i < n$. Transformation sequences in $\mathscr{G}$ form a category **Trafo**($\mathscr{G}$) with all TG-typed graphs as objects. Arrow composition is defined as sequential composition (that is, concatenation of sequences), and the identity of an object G is the empty sequence $G : G \Rightarrow G$.

Example 2.5 (Typed graph transformation). Figure 3 shows the three rules of a graph transformation system modelling an airline reservation system. The type graph of the system is given in Figure 1. The first rule describes the reservation of a seat by a customer. It is only applicable if the seat is listed as free. The result of the operation is recorded both by the customer and by the system by the creation of two edges of type rsrv and rsrvd, respectively. After the seat is reserved, it has to be acknowledged by the customer, which is modelled by the second rule. The third rule describes how the acknowledgment triggers the creation of an Invoice.

Figure 4 shows the span representation of the graph transformation rule rsrvSeat of Figure 3 as well as its application to an instance graph with two free seats in flight lh201. Operationally speaking, the application of the rule proceeds as follows. Given the occurrence d_L of the left-hand-side L in G that maps seat s with 1b, the application consists of two steps: the elements of G matched by $L \setminus l(K)$ are removed, which leads to the graph D without the free edge; then, the elements matched by $R \setminus r(K)$ are added to D, leading to the derived graph H.

Gluing the graphs L and D over their common part K yields again the given graph G, that is, D is a so-called *pushout complement* and the left-hand square (1) is a pushout square. Only in this case is the application permitted. Similarly, the derived graph H is the gluing of D and R over K, which forms the right-hand side pushout square (2).

The formalization of deletion as 'inverse gluing' implies that the application of a rule can be described as an inclusion of K into a context by means of the morphism $d_K : K \to D$ (*cf.* Figure 4). This implies that only vertices in the image of K can be merged or connected to edges in the context. These observations are reflected, respectively, in the *identification* and the *dangling condition* of the DPO approach, which characterize, given a rule $p : s = (L \overset{l}{\longleftarrow} K \overset{r}{\longrightarrow} R)$ and an occurrence $d_L : L \to G$ of the left-hand

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

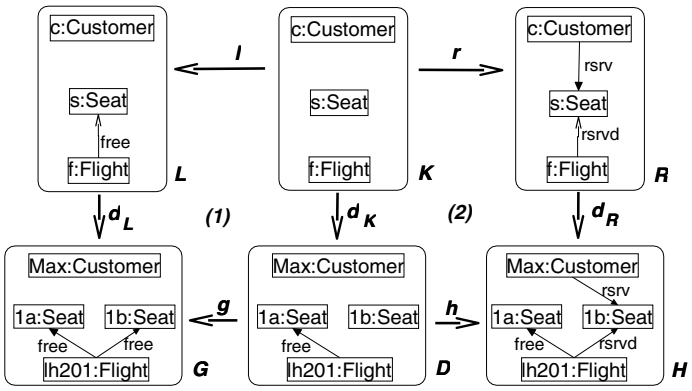


Fig. 4. DPO graph transformation step rsrvSeat(Max,1a) using rule rsrvSeat(c,s).

side, the existence of the pushout complement (1), and hence of a transformation step[†] $G \overset{p/d}{\Longrightarrow} H$. The *identification condition* states that objects from the left-hand side may only be identified by the match if they also belong to the interface (and are thus preserved). The *dangling condition* ensures that the structure D obtained by removing from G all objects that are to be deleted is indeed a graph, that is, no edges are left 'dangling' without source or target node.

2.3. *Independence and shift-equivalence of graph transformations*

Following our idea from the Introduction that graph transformation can be considered as a model for information (or database) systems, so far we have provided a model for database schemas as type graphs, database states as corresponding instance graphs, and database operations as graph transformation rules. In order to synchronize the concurrent access of different clients, it is necessary for the scheduler to analyze the conflicts and dependencies between these operations.

The theory of the DPO approach to graph transformation provides a semantic model for this analysis based on the notions of parallel and sequential independence and shift-equivalence (Kreowski 1977). A survey of this model and its more recent development towards a theory of graph process and event structures along the lines of Petri nets (Reisig and Goltz 1983; Winskel 1986) is given in Baldan *et al.* (1999).

Example 2.6 (Scheduling of transactions). In Figure 5, a sequence of transformations is depicted modelling the booking of seat 1b by customer Max. Throughout this paper we show such sequences by the given and derived graphs of the steps, recording the relevant part of the match in the parameters of the rule names. More abstractly, the same sequence can be denoted by the trace

bookSeat(Max,1b) = rsrvSeat(Max,1b); ackRsrv(Max,1b); issueInvoice(Max,1b)

[†] The pushout (2) always exists since category $\mathbf{Graph}_{\mathbf{TG}}$ is cocomplete due to the cocompleteness of **Graph**.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

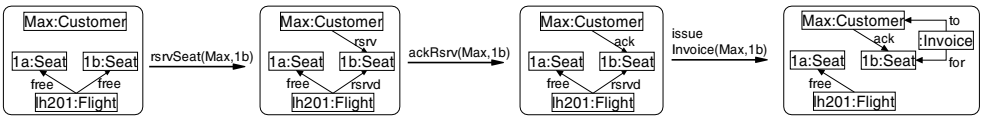


Fig. 5. DPO graph transformation sequence in the airline reservation system.

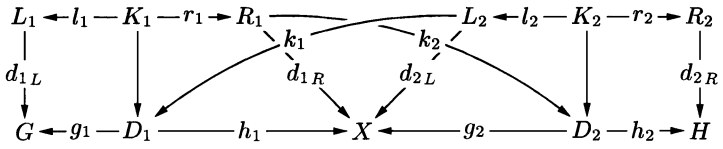


Fig. 6. Sequential independence.

The sequential booking of both seats in the first graph of the sequence would be represented as bookSeat(Max,1b) ; bookSeat(Max,1a) =

rsrvSeat(Max,1b); ackRsrv(Max,1b); issueInvoice(Max,1b);
rsrvSeat(Max,1a); ackRsrv(Max,1a); issueInvoice(Max,1a) (ii)

The two transactions are not in conflict because the commonly accessed objects are only in read-access. Thus they could be executed concurrently leading, for example, to the traces

rsrvSeat(Max,1b); ackRsrv(Max,1b); rsrvSeat(Max,1a);
issueInvoice(Max,1b); ackRsrv(Max,1a); issueInvoice(Max,1a) or (ii)
rsrvSeat(Max,1b); rsrvSeat(Max,1a); ackRsrv(Max,1b);
issueInvoice(Max,1b); ackRsrv(Max,1a); issueInvoice(Max,1a) (iii)

These represent different, but equivalent schedulings of the same concurrent execution of bookSeat(Max,1a) and bookSeat(Max,1b). In fact, if the two transactions are executed by a data base management system, the scheduler is free to choose any trace for execution that is equivalent to the sequential trace above (Bernstein *et al.* 1987).

The basic idea behind this equivalence is to identify computations which differ only in the scheduling of independent steps. Thus, we first introduce the notion of independence of DPO graph transformation steps.

Definition 2.7 (Sequential and parallel independence). We say a transformation sequence $G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$ as in Figure 6 is *sequentially independent* if there exist graph morphisms $R_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $g_2 \circ k_2 = d_{1R}$ and $h_1 \circ k_1 = d_{2L}$. Dually, two transformation steps $G \overset{p_1/d_1}{\Longrightarrow} X$ and $G \overset{p_2/d_2}{\Longrightarrow} Y$ as in Figure 7 are *parallel independent* if there exist graph morphisms $L_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $g_2 \circ k_2 = d_{1L}$ and $g_1 \circ k_1 = d_{2L}$.

Intuitively, two consecutive steps $G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$ are sequentially independent if they may be swapped, that is, if p_2 can be applied to G, and p_1 to the resulting graph. Moreover,

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

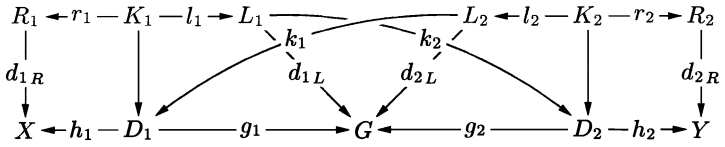


Fig. 7. Parallel independence.

two transformation steps $G \overset{p_1}{\Longrightarrow} X$ and $G \overset{p_2}{\Longrightarrow} Y$ are parallel independent if they are not in conflict, that is, either of them can be delayed after the other (see Corradini *et al.* (1997) for a recent survey on the theory of independence and concurrency in the DPO approach).

Theorem 2.8 (Local Church–Rosser). Given sequentially independent transformations $G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$, there exist a graph Y and a transformation step $G \overset{p_2/d_2'}{\Longrightarrow} Y$ such that $G \overset{p_1/d_1}{\Longrightarrow} X$ and $G \overset{p_2/d_2'}{\Longrightarrow} Y$ are parallel independent.

Given parallel independent transformation steps $G \overset{p_1/d_1}{\Longrightarrow} X$ and $G \overset{p_2/d_2}{\Longrightarrow} Y$, there exist a graph H and a transformation step $X \overset{p_2/d_2'}{\Longrightarrow} H$ such that $G \overset{p_1}{\Longrightarrow} X \overset{p_2}{\Longrightarrow} H$ is sequentially independent.

By combining the two statements of the above theorem, we can swap two sequentially independent steps $G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$ by first anticipating the application of p_2 and then delaying the application of p_1. This construction defines an equivalence on transformation sequences as the least congruence with respect to sequential composition and identities identifying $G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$ and $G \overset{p_2/d_2'}{\Longrightarrow} Y \overset{p_1/d_1'}{\Longrightarrow} H$ whenever the second sequence is obtained from the first by swapping the two steps.

Example 2.9 (Equivalent transformation sequences). Figure 8 exemplifies in more detail the equivalence of the traces (i), (ii) and (iii) of Example 2.6. It is shown how the traces can be transformed into each other by permuting the respective second, third and fourth steps. In each square, representing the exchange of two steps, the top and the left steps are parallel independent. Thus they can be delayed leading to the bottom, and the right steps, respectively. Moreover, the sequences given by the top-right steps are sequentially independent, which means that they can be swapped, leading to the left-bottom sequences. The reason is that only customer Max and flight lh201 are shared between the steps, and they are in read-access only.

In contrast with the presentation in Kreowski (1977), the following definition of *shift*-equivalence is restricted to sequential transformation sequences. This is justified by our interest in abstract semantics, instead of its (canonical) representation. In fact, as shown in Kreowski (1977), all parallel DPO sequences can be transformed into shift-equivalent sequential ones. Thus, shift-equivalence classes of parallel sequences are one-to-one with shift-equivalence classes of purely sequential transformations. The main benefit of parallel transformations is that they provide a canonical representation of shift-equivalence classes (Kreowski 1977), which, however, is of no concern here.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

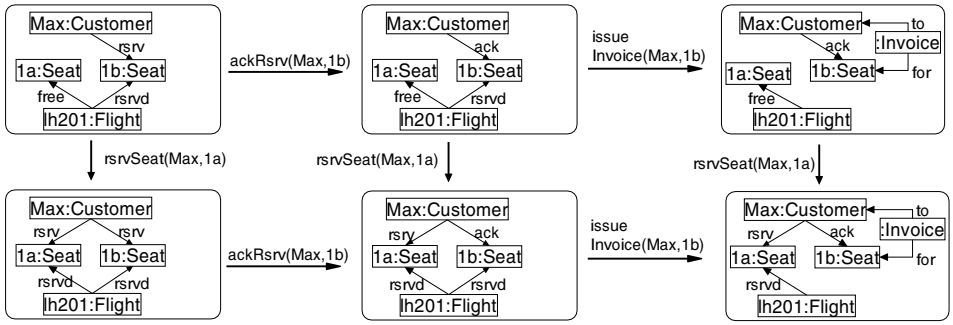


Fig. 8. Independence and shift-equivalence.

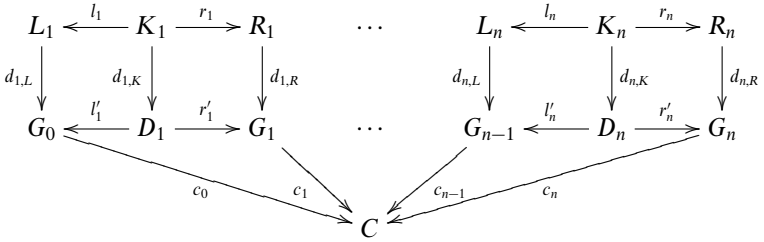
Definition 2.10 (Shift-equivalence on sequential transformations). We say two transformations $\sigma = G \overset{p_1/d_1}{\Longrightarrow} X \overset{p_2/d_2}{\Longrightarrow} H$ and $\sigma' = G \overset{p_2/d_2'}{\Longrightarrow} Y \overset{p_1/d_1'}{\Longrightarrow} H$ are in *swap-correspondence* if they are sequentially independent and σ' is the result of exchanging in σ the first and the second step according to Theorem 2.8.

Shift-equivalence $\equiv_{sh}$ is the least equivalence relation on transformations including swap-correspondence, and that is closed under sequential composition of transformations. The *category of concurrent transformation sequences in* $\mathcal{G}$ is the quotient $\mathbf{Trafo}(\mathcal{G})/_{sh}$ of $\mathbf{Trafo}(\mathcal{G})$ with respect to the *shift*-equivalence.

The category $\mathbf{Trafo}(\mathcal{G})/_{sh}$ is well defined since *shift*-equivalence does not identify objects.

In Example 2.6 we spoke of *concurrent transactions*, that is, sequences whose individual steps are mutually independent. In order to make this notion precise, we have to extend the definition of independence of DPO steps towards DPO sequences.

Definition 2.11 (Independence of DPO sequences). Let $\sigma : G_0 \Rightarrow G_n$ be a transformation sequence and let $\langle c_i, C \rangle$ be the colimit of the corresponding double pushout diagrams as below.



Two double pushout rewriting steps p_i/d_i and p_j/d_j, with $1 \leqslant i < j \leqslant n$, are called *independent* if there exist graph morphisms $k_i : L_j \to D_i$ and $k_j : R_i \to D_j$ such that (with the notation of the diagram above) $c_i \circ r_i' \circ k_i = c_{j-1} \circ d_{j,L}$ and $c_{j-1} \circ l_j' \circ k_j = c_i \circ d_{i,R}$.

Let $\sigma = \sigma_1 ; \sigma_2$ be a transformation sequence with $\sigma_1 : G_0 \Rightarrow G_k$ and $\sigma_2 : G_k \Rightarrow G_n$. The transformation sequences σ_1 and σ_2 are called *sequentially independent* if for every $1 \leqslant i \leqslant k$ and $k < j \leqslant n$, the double pushout transformation steps p_i/d_i and p_j/d_j are independent.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

The colimit construction used above for defining the independence of non-consecutive steps in a DPO sequence has its root in the theory of graph processes (Corradini *et al.* 1996), an alternative presentation of shift-equivalence classes of DPO sequences (see, for example, Baldan *et al.* (1999) for a survey). The idea is to construct a global graph C as the union of all the graphs in the sequence such that two elements (nodes or edges) of two graphs G_i and G_j are merged in C if and only if they represent the same element in two different states of the computation.

In order to justify our notion of independence of sequences, we prove a local Church–Rosser-like theorem, which states that independent sequences can be swapped.

Theorem 2.12 (Swapping independent DPO sequences). Let $\sigma = \sigma_1; \sigma_2$ be a DPO sequence such that σ_1 and σ_2 are sequentially independent. Then, with the notation of Definition 2.11, there exists a sequence $\sigma' = \sigma'_1; \sigma'_2$ with $\sigma'_1 = p_{k+1}/d'_{k+1}; \ldots; p_n/d'_n : G_0 \Rightarrow G'_{n-k}$ and $\sigma'_2 = p_1/d'_1; \ldots; p_k/d'_k : G'_{n-k} \Rightarrow G_n$, such that σ and σ' are shift equivalent.

Proof. First, observe that the colimit injections c_i in Definition 2.11 are injective. In fact, up to isomorphism, the graph C can be regarded as the union of the graphs $G_0, \ldots, G_n$, sharing the interface graphs $D_1, \ldots, D_n$. Therefore, the independence of two steps p_i/d_i and p_j/d_j specializes to the classical notion of sequential independence (*cf.* Definition 2.7) if the two steps are consecutive, that is, $j = i+1$. Thus, they can be swapped leaving two independent steps in the sense of both Definition 2.7 and 2.11. Moreover, it follows from Definition 2.10 of shift-equivalence that the transformations before and after the exchange are shift-equivalent. From the equivalence of graph processes and shift-equivalence classes (*cf.* Baldan *et al.* (1999)), this implies that the new DPO sequence yields, up to isomorphism, the same process. Thus, the (in)dependencies between the steps involved in the exchange and the remaining steps in the sequence do not change.

This allows us to exchange, iteratively, the first step p_{k+1}/d_{k+1} of σ_2 with all steps of σ_1, which are all independent of p_{k+1}/d_{k+1} by assumption. Repeating this with the remaining steps of σ_2, we finally obtain the sequence σ', which is related to σ only be a series of swap operations, and is, therefore, shift-equivalent. □

This concludes our presentation of the theory of DPO graph transformation systems as a model for information systems covering the concepts of schema and instance, operation, and concurrency. Next, we are going to extend this model by the notion of view.

3. Views and open graph transformation systems

In this section, we define a notion of *view of a graph transformation systems*, that is, a subsystem with incomplete information about the data and operations of the bigger system. As semantics of views, *graph transitions* are introduced, which generalize DPO steps by allowing under-specified (or completely unspecified) steps. Graph transitions, which are characterized by means of a double-pullback construction, can be seen as a loose semantics of graph transformation systems.

However, this loose semantics is often too liberal to ensure a reasonable behaviour. Thus, graph transformation systems are extended with an interface description in terms

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

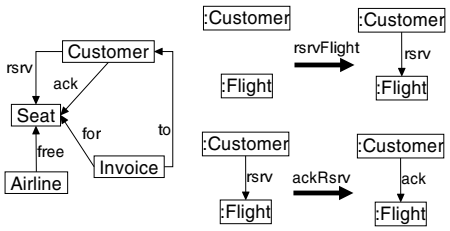


Fig. 9. The customer view of the airline reservation system.

of *open types* restricting the interaction of the system with its context. This leads to the notion of an *open graph transformation system.*

3.1. *Views and loose semantics*

In information systems, we distinguish between the integrated *system view* and one or several *client views*, which represent projections of the system view to the data and operations that should be accessible by a particular group of users or a certain client application. The model of a client view by a graph transformation system is, therefore, a partial one compared to the complete system model.

Example 3.1 (Views). Consider the graph transformation system in Figure 9, which represents the view of a customer of the airline reservation system. Notice, that customers can observe the free seats in a flight, but it is not their responsibility to make sure that only free seats are reserved. The list of reserved seats in a flight (represented by edge type rsrvd) is hidden from a customer's view, as well as the operation issueInvoice, which is only invoked by the system.

Thus, the type graph of the graph transformation system modelling the customer's view is a subgraph of the complete type graph in Figure 1, one rule is missing, and another one is reduced in the smaller view.

Formally, the notion of view is based on the reduction of graphs, rules and transformations induced by a subgraph TG_1 of the type graph TG of a graph transformation system: this defines a functor $_|_{TG_1} : \mathbf{Graph}_{TG} \to \mathbf{Graph}_{TG_1}$ as follows. For $G \in \mathbf{Graph}_{TG}$, the projection $G|_{TG_1}$ is defined as the largest subgraph of G that can be typed over TG_1, formally a pullback (inverse image) of the typing homomorphism $g : G \to TG$ of an instance graph $\langle G, g \rangle$ and the inclusion $TG_1 \hookrightarrow TG$. Analogously, we define the projection of graph morphisms, rule spans, as well as DPO diagrams. It is shown, for example, in Heckel *et al.* (1996) that this indeed defines a functor that preserves pushouts and pullbacks. The projection of a DPO diagram is depicted in Figure 10, where the two back squares are a DPO diagram in $\mathbf{Graph}_{TG}$ and the two front squares form a DPO diagram in $\mathbf{Graph}_{TG_1}$. The commutative double cube is constructed by projecting the corresponding graphs and morphisms; that is, computing the corresponding horizontal pullback squares.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

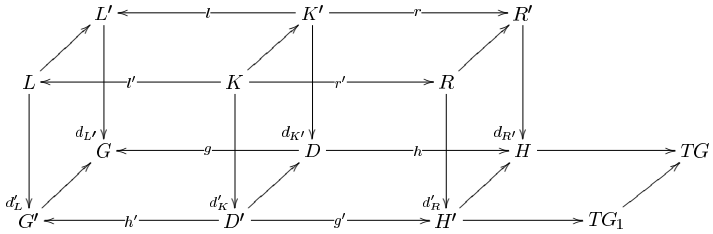


Fig. 10. DPO diagram projection.

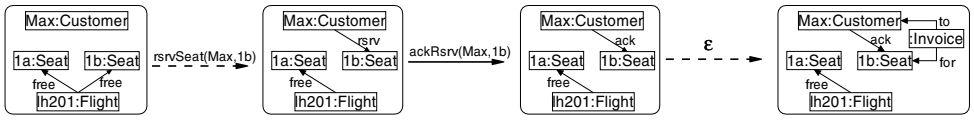


Fig. 11. The customer view of the transformation sequence in the airline reservation system.

Based on this projection, we define a notion of view of a system capturing the intuition of the above example.

Definition 3.2 (Subrule and view). Given two rule spans $s = (L \xleftarrow{l} K \xrightarrow{r} R)$ and $s' = (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$, we say that s is a subrule of s', written $s \subseteq s'$, if $L \subseteq L'$, $K \subseteq K'$, and $R \subseteq R'$ such that the two squares (1) and (2) below are pullbacks.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ \downarrow & (1) & \downarrow & (2) & \downarrow \\ L' & \xleftarrow[l']{} & K' & \xrightarrow[r']{} & R' \end{array}$$

A graph transformation system $\mathcal{G}_1 = \langle TG_1, P_1, \pi_1 \rangle$ is called a *view* of another system $\mathcal{G} = \langle TG, P, \pi \rangle$ if $TG_1 \subseteq TG$ and $P_1 \subseteq P$ such that, for all $p \in P_1$, $\pi_1(p) \subseteq \pi(p)|_{TG_1}$.

The intuition behind the notion of subrule in the definition above is that the bigger rule span s' deletes, preserves and creates at least as much as the smaller one s. In particular, the fact that the two squares are pullbacks ensures that a structure that is deleted and re-generated by s cannot be part of the interface of s'.

Example 3.3 (Projection of behaviour). Figure 11 shows the customer's view of the transformation sequence in the airline reservation system shown in Figure 5. The graphs are obtained as projections of the graphs of the transformation sequence to the smaller type graph. The three steps represent three different cases. The first step is no longer a DPO transformation since the effect of removing the free edge is not specified by the rule rsrvSeat in Figure 9. The second step models a local operation of the customer, and it remains a DPO step in the projection. The third step appears in the customer's view as a spontaneous state transition because the rule issueInvoice is completely hidden.

In order to provide views with a semantics independent of the complete system model, a more liberal notion of graph transformation is required that allows us to capture underspecified as well as spontaneous steps. With this motivation, in Heckel *et al.* (1999) and

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

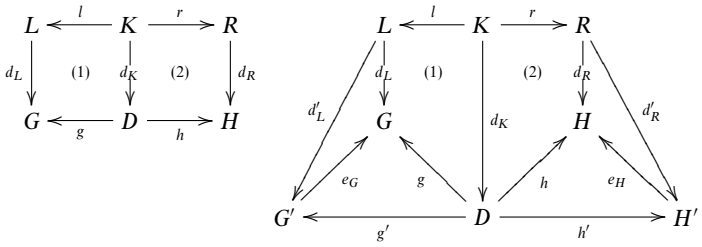


Fig. 12. DPB transition as DPO transformation plus change of context.

Heckel *et al.* (2001) the notion of *graph transition* has been introduced. Graph transitions are defined by replacing the double-pushout diagram of a transformation step with a double-*pullback* (DPB)[†].

Definition 3.4 (Graph transition). Let $p : s$ be a rule span with $s = (L \xleftarrow{l} K \xrightarrow{r} R)$. Then, a *graph transition* from G to H via p, denoted by $G \overset{p/d}{\rightsquigarrow} H$, is a diagram like the left part of Figure 12 where both (1) and (2) are pullback squares. A graph transition (or transition, for short) is called *injective* if both g and h are injective graph morphisms. It is called *faithful* if it is injective and the morphisms d_L and d_R satisfy the following condition; for all $x, y \in L$, $y \notin l(K)$ implies $d_L(x) = d_L(y)$, and analogously for d_R.

The last condition means that d_L and d_R satisfy the identification condition of the DPO approach (Corradini *et al.* 1997) with respect to l and r.

Notice that any pushout square in $\mathbf{Graph}_{TG}$ of two given morphisms such that one of them is injective is also a pullback square. Thus, every DPO transformation is also a DPB transition.

Figure 13 shows in detail a (faithful) graph transition using rsrvSeat. Notice that the left square is not a pushout because of the additional free edge in G, which disappears in D without being explicitly deleted.

Each faithful transition can be regarded as a transformation step plus a change-of-context (Heckel 1998; Heckel *et al.* 2001). This is modelled by additional deletion and creation of elements before and after the actual step, both represented by injective graph morphisms. The transition in Figure 13, for example, could be represented as a DPO transformation preceded by an additional step removing the free edge.

Theorem 3.5 (Transitions as transformation step plus change-of-context). If $p : s$ is a rule span, the following statements are equivalent (*cf.* Figure 12).

(i) There is a faithful transition $G \overset{p/d}{\rightsquigarrow} H$ with $d = \langle d_L, d_K, d_R \rangle$ as in Figure 12 on the left.

[†] This should not be confused with the pullback approach (Bauderon 1996; Bauderon and Jacquet 2001), which has been proposed as a categorical formulation of the Node Label Controlled (NLC) approach to graph transformation (Engelfriet and Rozenberg 1997).

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

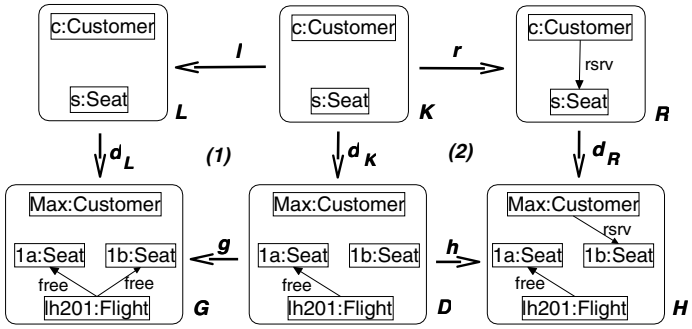


Fig. 13. A sample DPB graph transition.

(ii) There are a transformation step $G' \overset{p/d'}{\Longrightarrow} H'$ and injective graph morphisms $e_G : G' \to G$ and $e_H : H' \to H$ such that $e_G \circ g' = g$ and $e_H \circ h' = h$ (see Figure 12 on the right).

Faithful transitions capture our intuition about transitions as a loose interpretation of graph transformation rules. Unless otherwise stated, all transitions in the following are assumed to be faithful.

Transition sequences are defined analogously to transformation sequences in Definition 2.4. The resulting category of graphs and (faithful) transition sequences in a graph transformation system $\mathscr{G}$ is denoted by $\mathbf{Trans}(\mathscr{G})$.

In order to allow state transformations by operations of the environment when the subsystem under consideration is idle, we introduce the ϵ-rule $\epsilon : \varnothing \leftarrow \varnothing \to \varnothing$. Every injective span of morphisms $s = (G \leftarrow D \to H)$ can be turned into a (faithful) ϵ-transition, denoted by $G \overset{\epsilon/s}{\rightsquigarrow} H$. Given a view $\mathscr{G}_1$ of $\mathscr{G}$, each transition sequence in $\mathscr{G}$ can now be projected onto a sequence of transitions in $\mathscr{G}_1$.

Proposition 3.6 (Projection of sequential behaviour). Assume graph transformation systems $\mathscr{G}_1 = \langle TG_1, P_1, \pi_1 \rangle$ and $\mathscr{G} = \langle TG, P, \pi \rangle$ such that $\mathscr{G}_1$ is a view of $\mathscr{G}$, and let $\mathscr{G}_1$ contain the ϵ-rule. Then, there exists a functor $_|_{\mathscr{G}_1} : \mathbf{Trans}(\mathscr{G}) \to \mathbf{Trans}(\mathscr{G}_1)$ defined on transitions by

$$G \overset{p/d}{\rightsquigarrow} H \quad \mapsto \quad G|_{TG_1} \overset{p/(d|_{TG_1})}{\rightsquigarrow} H|_{TG_1} \qquad \text{if } p \in P_1$$
$$G \overset{p/d}{\rightsquigarrow} H \quad \mapsto \quad G|_{TG_1} \overset{\epsilon/s|_{TG_1}}{\rightsquigarrow} H|_{TG_1} \qquad \text{if } p \notin P_1$$

where $s = G \leftarrow D \to H$ denotes the bottom span of DPB diagram d (see Figure 10 with all squares now pullback squares). The functor extends freely to transition sequences.

Proof. The mapping defined above is well defined since the image by $_|_{\mathscr{G}_1}$ of a DPB diagram is a DPB diagram. Moreover, it is total because every span of injective morphisms $s = (G \leftarrow D \to H)$ is a graph transition from $G|_{\mathscr{G}_1}$ to $H|_{\mathscr{G}_1}$ via the ϵ-rule. Finally, since composition of morphisms in $\mathbf{Trans}(\mathscr{G})$ and $\mathbf{Trans}(\mathscr{G}_1)$ is defined by sequential composition of transitions it is clear that it indeed yields a functor. □

An example of such a projection is the sequence in Figure 11, which represents the customer's view of the transformation sequence in Figure 5 (*cf.* Example 3.3).

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

3.2. *Open graph transformation systems*

Double-pushout graph transformation, like most rewriting approaches, come with an implicit frame condition stating that nothing should change in the current state beyond what is explicitly specified by the rule. The notion of double-pullback graph transition disregards this frame condition allowing arbitrary changes as long as they do not conflict with the changes specified in the rule. In general, this notion of transition is too liberal for approximating the semantics of a view of a bigger system. Therefore, in this section we introduce the possiblity of specifying *explicit frame conditions*, which limit unspecified changes to the instances of certain *open types*, that is, vertex and edge types acting as interface with the enclosing system.

Definition 3.7 (Explicit frame condition). We use $\cup TG = TG_V \cup TG_E$ to denote the (disjoint) union of the node and edge types in a type graph TG. An *explicit frame condition* over a type graph TG is a pair of sets $O = \langle O^-, O^+ \rangle$ of *output* and *input open types* $O^-, O^+ \subseteq \cup TG$.

A type $t \in \cup TG$ is called *output (input) closed* if $t \notin O^-$ ($t \notin O^+$). A graph transition $G \overset{p/d}{\rightsquigarrow} H$ *satisfies* $O^-(O^+)$ if for every output (input) closed type t, the reduct of the left-hand side (right-hand side) square of DPB diagram d (*cf.* Figure 2 on the right) to the type t forms a pushout in **Set**. The transition satisfies $O = \langle O^-, O^+ \rangle$ if it satisfies both O^- and O^+.

Formally, for each $t \in \cup TG$, the reduct forms a functor $|_|_t : \mathbf{Graph}_{TG} \to \mathbf{Set}$. Because of the component-wise construction of $\mathbf{Graph}_{TG}$-pushouts in **Set**, functor $|_|_t$ preserves pushouts. Thus, each DPO step satisfies the most restrictive frame condition $O = \langle \varnothing, \varnothing \rangle$. Moreover, DPB transitions satisfying this frame conditions are necessarily DPO transformations.

An open graph transformation $\mathcal{O}$ is defined as a graph transformation system $\mathcal{G}$, together with an explicit frame condition O. Transitions in $\mathcal{O}$ are just transitions in $\mathcal{G}$ that satisfy the frame condition O. Moreover, we assume the presence of ϵ-rules.

Definition 3.8 (Open graph transformation system). An *open graph transformation system* $\mathcal{O} = \langle TG, P, \pi, O \rangle$, briefly OGS, consists of a graph transformation system $\mathcal{G} = \langle TG, P, \pi \rangle$ containing the ϵ-rule, and an explicit frame condition $O = \langle O^-, O^+ \rangle$ over TG. A transition in $\mathcal{O}$ using rule $p : s$ is a transition in $\mathcal{G}$ satisfying the frame condition O. The corresponding category of transition sequences in $\mathcal{O}$ is denoted by $\mathbf{Trans}(\mathcal{O})$.

An open graph transformation system $\mathcal{O}_1 = \langle \mathcal{G}_1, O_1 \rangle$ is called a *view of* $\mathcal{O} = \langle \mathcal{G}, O \rangle$ if $\mathcal{G}_1$ is a view of $\mathcal{G}$ and:

(i) For each rule $p \in P \setminus P_1$ with $\pi(p) = s = (L \leftarrow K \rightarrow R)$, the ϵ-transition $L|_{TG_1} \overset{\epsilon/s|_{TG_1}}{\rightsquigarrow} R_{TG_1}$ satisfies the frame condition O_1.
(ii) For each rule $p \in P_1$, the transition p/e where e is the subrule diagram of Definition 3.2 satisfies the frame condition O_1.
(iii) The frame conditions are strengthened, that is, $O^- \cap TG_1 \subseteq O_1^-$ and $O^+ \cap TG_1 \subseteq O_1^+$.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

Spelling out condition (i) above, the TG_1-reduct of the left-hand side of the rule span s has to be an isomorphism for every output closed type $t \in \cup TG_1 \setminus O_1^-$. Analogously, the right-hand side injection of $s|_{TG_1}$ has to be an isomorphism on each input closed type $t \in \cup TG_1 \setminus O_1^+$. This means that rules that are newly introduced in the bigger system must not affect the closed types of the view.

Condition (ii) requires that the TG_1-reduct of the left-hand side square of the subrule diagram is a pushout when restricted to output closed types and, analogously, the TG_1-reduct of the right-hand side square is a pushout when restricted to creation-closed types. This is similar to condition (i), except that it applies to rule extensions rather than to newly introduced rules. Together, (i) and (ii) ensure that extended or newly introduced rules of $\mathscr{O}$ can be seen as transitions in $\mathscr{O}_1$.

The last condition, (iii), ensures that valid transitions in $\mathscr{O}$ remain valid transitions in $\mathscr{O}_1$ under the projection. This is ensured by requiring that the frame conditions in $\mathscr{O}$ are more restrictive than in $\mathscr{O}_1$, that is, all types of TG_1 that were open in $\mathscr{O}$ remain open in $\mathscr{O}_1$.

The graph transformation system for the customer's view in Figure 9 is turned into an open graph transformation system by defining $O^- = \{\mathsf{free}\}$ and $O^+ = \{\mathsf{Invoice}, \mathsf{from}, \mathsf{to}\}$. The transition sequence in Figure 11 satisfies the constraints imposed by this frame condition, since the only unspecified effects are the removal of the free edge and the creation of the Invoice.

Proposition 3.9 (Projection of sequential behaviour with open types). We assume open graph transformation systems $\mathscr{O}_1$ and $\mathscr{O}$, as above, such that $\mathscr{O}_1$ is a view of $\mathscr{O}$. Then, the functor $_|_{\mathscr{G}_1}$ of Proposition 3.6 induces a functor $_|_{\mathscr{O}_1} : \mathbf{Trans}(\mathscr{O}) \to \mathbf{Trans}(\mathscr{O}_1)$, defined by $\tau|_{\mathscr{O}_1} = \tau|_{\mathscr{G}_1}$.

Proof. Since we have shown that $_|_{\mathscr{G}_1}$ is a functor between $\mathbf{Trans}(\mathscr{G})$ and $\mathbf{Trans}(\mathscr{G}_1)$, it remains to prove that $\tau|_{\mathscr{O}_1} = \tau|_{\mathscr{G}_1}$ is well defined, that is, the image of any transition in $\mathscr{O}$ satisfies the frame condition $\mathscr{O}_1$. Let $G \overset{p/d}{\rightsquigarrow} H$ be a transition in $\mathscr{O}$. If p does not belong to P_1, conditions (i) and (iii) in the definition above, together with the fact that projections preserve pushouts, entail that its image satisfies the frame condition $\mathscr{O}_1$. If p is a rule of P_1, the image of $G \overset{p/d}{\rightsquigarrow} H$ by $\tau|_{\mathscr{O}_1}$ is a transition $G|_{TG_1} \overset{p/(d|_{TG_1})}{\rightsquigarrow} H|_{TG_1}$ in $\mathscr{G}_1$ constructed as in Figure 10. Again, the fact that projections preserve pushouts and pullbacks, together with conditions (ii) and (iii) in the definition above, entail that the transition $G|_{TG_1} \overset{p/(d|_{TG_1})}{\rightsquigarrow} H|_{TG_1}$ satisfies the frame condition $\mathscr{O}_1$, which concludes the proof. □

In this section we have provided a model for views of information systems based on graph transformation systems. However, the model is far from complete because database views may be defined via complex queries based, for example, on the structure of instances and the concrete values of attributes, while our notion of view uses only projection along types. On the other hand, our model goes further than the notion of database view by taking into account (sequential) behaviour. In the next section, this will be extended towards the concurrent behaviour of views.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

4. Independence and shift-equivalence of graph transitions

In this section, we first generalize parallel and sequential independence from transformations to graph transitions. Then, we prove a local Church–Rosser theorem and derive a corresponding notion of shift-equivalence.

We follow the same intuition as in Section 2.3 to formalize a notion of conflict that allows concurrent read but exclusive write access. In fact, the definition below is just an application of the same conditions to the more general notion of transition.

Definition 4.1 (Sequential and parallel independence). A sequence of two transitions $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ as in Figure 6 is called *sequentially independent* if there exist graph morphisms $R_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $g_2 \circ k_2 = d_{1R}$ and $h_1 \circ k_1 = d_{2L}$.

Dually, two transitions $G \overset{p_1/d_1}{\rightsquigarrow} X$ as in Figure 7 and $G \overset{p_2/d_2'}{\rightsquigarrow} Y$ are *parallel independent* if there exist graph morphisms $L_1 \xrightarrow{k_2} D_2$ and $L_2 \xrightarrow{k_1} D_1$ such that $g_2 \circ k_2 = d_{1L}$ and $g_1 \circ k_1 = d_{2L}$.

Notice that, according to the definition above, only such conflicts are excluded that involve explicitly specified effects of at least one of the two steps. Thus, we are not interested in the interference between the (for us invisible) operations of the environment. Notice also that, because of unspecified effects, the classic characterization of independence in terms of the intersection of matches (and co-matches) does not hold anymore: This would only capture the absence of internal conflicts, that is, between operations of the same view.

As in the DPO approach, the sequential independence of two transitions $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ means that they can be swapped to $G \overset{p_2/d_2'}{\rightsquigarrow} Y \overset{p_1/d_1'}{\rightsquigarrow} H$. However, two parallel independent transitions $G \overset{p_1/d_1}{\rightsquigarrow} X$ and $G \overset{p_2/d_2'}{\rightsquigarrow} Y$ may not be sequentialized to $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ as for the Church–Rosser theorem in the DPO approach.

Theorem 4.2 (Local Church–Rosser). Assume an open graph transformation system $\mathcal{O} = \langle \mathcal{G}, O \rangle$ and let p_1, p_2 be rule names in P.

Given sequentially independent transitions $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ in $\mathcal{O}$, there exist a graph Y and a transition $G \overset{p_2/d_2'}{\rightsquigarrow} Y \overset{p_1/d_1'}{\rightsquigarrow} X$ sequentially independent in $\mathcal{O}$ such that $G \overset{p_1/d_1}{\rightsquigarrow} X$ and $G \overset{p_2/d_2'}{\rightsquigarrow} Y$ are parallel independent.

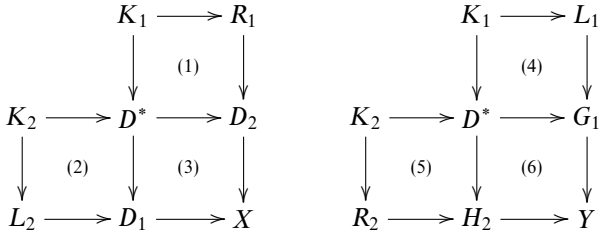
Let $O = \langle O^-, O^+ \rangle$ with $O^- = O^+$. In this case, given parallel independent transitions $G \overset{p_1/d_1}{\rightsquigarrow} X$ and $G \overset{p_2/d_2'}{\rightsquigarrow} Y$ in $\mathcal{O}$, there exist a graph H and transitions $X \overset{p_2/d_2}{\rightsquigarrow} H$ and $Y \overset{p_1/d_1'}{\rightsquigarrow} H$ in $\mathcal{O}$ such that $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ and $G \overset{p_2/d_2'}{\rightsquigarrow} Y \overset{p_1/d_1'}{\rightsquigarrow} H$ are sequentially independent.

Proof. As in the DPO case, the second part of the local Church–Rosser theorem follows from the first because, by inverting $G \overset{p_1}{\rightsquigarrow} H_1$, we obtain an independent sequence $H_1 \overset{p_1^{-1}/d_1^{-1}}{\rightsquigarrow} G \overset{p_2/d_2}{\rightsquigarrow} H_2$. However, while DPO steps are always invertible, this is not true for DPB transitions with frame conditions since, in general, different restrictions are imposed on the left and the right-hand side pullbacks. Therefore, condition $O^- = O^+$ above is

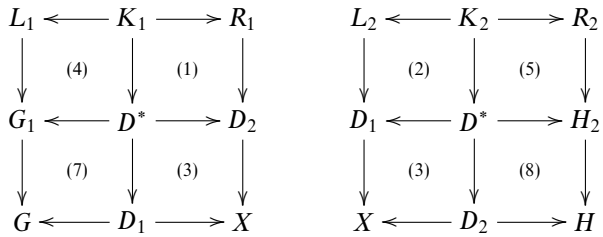
https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

required in order to ensure that transitions are invertible, and in this case the second part is a consequence of the first.

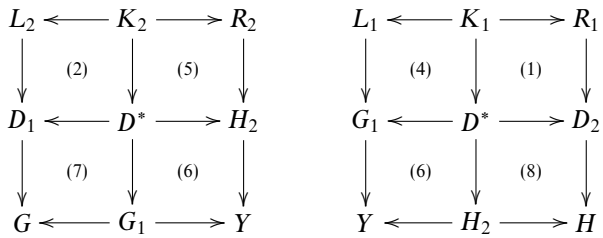
Let us prove, then, the first part of the theorem. Let $G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ be sequentially independent transitions in $\mathcal{O}$ (*cf.* Figure 6). We can build the commutative diagrams below in the following way. First, we make the pullback of the morphisms $D_1 \to X$ and $D_2 \to X$ such that the diagram on the left is a commutative diagram and all squares in it become pullback squares. Second, we build the diagram on the right making the corresponding pushout squares. Since injective morphisms are inherited under pushouts, the pushout squares in the diagram on the right are also pullback squares.



Combining these squares together with the pullback squares that provide the DPB-transitions, we can build the commutative diagrams below, where all squares except squares (7) and (8) are pullback squares.



However, since squares (4) and (5) above are also pushout squares, the squares (4+7) and (5+8) are pullbacks, and the horizontal morphisms are injective, we also get that squares (7) and (8) above are pullback squares with all their morphisms injective. Putting these pullbacks together, we obtain DPB-transitions



such that the DPB-transition sequence $G \overset{p_2}{\rightsquigarrow} Y \overset{p_1}{\rightsquigarrow} H$ is sequentially independent. Moreover, the joint surjectivity of pushouts, the pullback construction and the faithfulness of the initial DPB-transitions guarantee that the DPB-transitions constructed above are faithful.

Finally, it just remains to prove that these DPB-transitions satisfy the frame condition.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

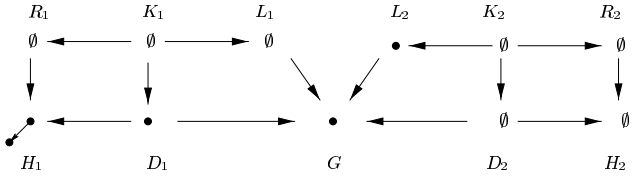


Fig. 14. Church–Rosser counterexample

Since squares (4), (5) and (6) above are pushout squares in $\mathbf{Graph}_{TG}$, it is clear that square (5)+(6) satisfies O^+ and square (4)+(6) satisfies O^-. Moreover, since square (2)+(3) restricted to nodes or edges of type $t \in \cup TG \setminus O^-$ is a pushout in **Set**, and square (3) is a pullback square in $\mathbf{Graph}_{TG}$ with all its morphisms injective, we obtain that square (2) restricted to nodes or edges of type $t \in \cup TG \setminus O^-$ is a pushout in **Set**. Furthermore, since square (4)+(7) satisfies O^- and (4) is a pushout square in $\mathbf{Graph}_{TG}$, we get that (7) satisfies O^-, which entails that (2)+(7) satisfies O^-. In a similar way we also get that square (1)+(8) satisfies O^+, which ends the proof. □

Notice that the above theorem does not hold in general if transitions are not assumed to be faithful: for general (in particular, non-injective) transitions it is not true that the parallel or sequential independence of the given transitions implies the applicability of the rules in the reverse order (Ehrig *et al.* 1999b). Moreover, if transitions are injective but not faithful, injectivity is not preserved in general. Thus, faithful transitions are the most general class that ensures that independent transitions can be exchanged, and that is stable under this operation.

As already noticed in the proof of Theorem 4.2, the second part of the local Church–Rosser theorem is a consequence of the first as long as the frame conditions are symmetric, that is, $O^- = O^+$. Otherwise, the reversibility of transitions is lost. As a consequence, two parallel independent transitions may be sequentialized, but we cannot guarantee that such a sequence of transitions satisfies the frame condition, as the following example shows.

Example 4.3. Let p_1 be the empty rule and p_2 be a rule deleting a single vertex of type • as depicted in Figure 14. Set $O^+ = TG$ and $O^- = \varnothing$. Consider the DPB transitions $G \overset{p_1/d_1}{\rightsquigarrow} H_1$ and $G \overset{p_2/d_2}{\rightsquigarrow} H_2$, where H_1 is a graph consisting of two nodes and one edge, $H_2 = D_2$ are empty graphs, and $G = D_1$ are graphs with only one node, with the obvious morphisms as in Figure 14. It is clear that the transitions are parallel independent and they satisfy the frame condition, but there is no DPB transition $H_1 \overset{p_2/d_2'}{\rightsquigarrow} X$ satisfying the frame condition, since the pushout in **Set** of $K_2 \to L_2$ and $L_2 \to H_1$ is not a graph because it has a dangling edge.

In analogy with Definition 2.10 of shift-equivalence for DPO sequences, we next define a corresponding notion for transition sequences based on the local Church–Rosser theorem above.

Definition 4.4 (Shift-equivalence on transition sequences). Two transition sequences $\tau = G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ and $\tau' = G \overset{p_2/d_2'}{\rightsquigarrow} Y \overset{p_1/d_1'}{\rightsquigarrow} H$ are in *swap-correspondence* if they are

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

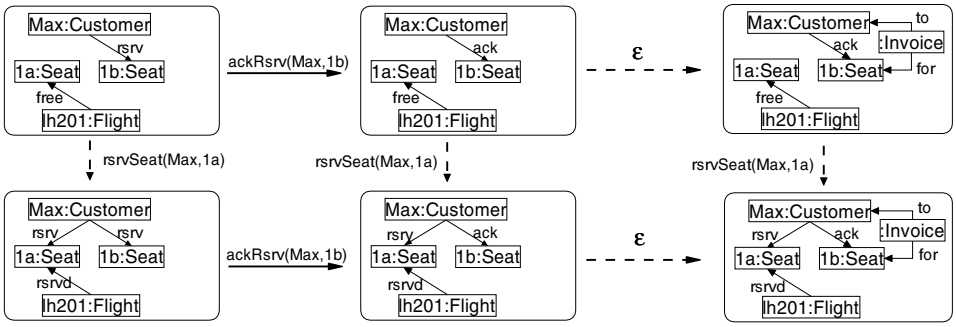


Fig. 15. Independence and shift-equivalence.

sequentially independent and τ' is the result of exchanging in τ the first and the second step according to Theorem 4.2.

Shift-equivalence $\equiv_{sh}$ is the least equivalence relation on transition sequences including swap-correspondence, and that is closed under sequential composition of sequences. The *category of* shift-*concurrent transition sequences in* $\mathscr{O}$ is the quotient $\mathbf{Trans}(\mathscr{O})/_{sh}$ of $\mathbf{Trans}(\mathscr{O})$ with respect to the *shift*-equivalence.

The category $\mathbf{Trans}(\mathscr{O})/_{sh}$ is well defined since *shift*-equivalence does not identify objects.

Example 4.5 (Shift-equivalent transition sequences). Figure 15 exemplifies the shift-equivalence of transition sequences. The transitions are projections of the DPO steps in Figure 8. It is worth noticing two different kinds of concurrency in this example. The left square represents what could be called *intra-concurrency*, that is, concurrency within the subsystem under consideration. The right square results from the concurrency between the subsystem and its environment, referred to as *inter-concurrency*.

Given a view $\mathscr{O}_1$ of an open graph transformation system $\mathscr{O}$, *shift*-concurrent transition sequences in $\mathscr{O}$ can be projected onto *shift*-concurrent transition sequences in $\mathscr{O}_1$. The following proposition states this fact.

Proposition 4.6 (Projection of concurrent behaviour). Assume open graph transformation systems $\mathscr{O}_1 = \langle \mathscr{G}_1, O_1 \rangle$ and $\mathscr{O} = \langle \mathscr{G}, O \rangle$ such that $\mathscr{O}_1$ is a view of $\mathscr{O}$. Then, there exists a functor $_||_{\mathscr{O}_1} : \mathbf{Trans}(\mathscr{O})/_{sh} \to \mathbf{Trans}(\mathscr{O}_1)/_{sh}$ defined by

$$[\tau]_{sh}||_{\mathscr{O}_1} = [\tau|_{O_1}]_{sh}.$$

Proof. It is straightforward to check that if two transition sequences $\tau = G \overset{p_1/d_1}{\rightsquigarrow} X \overset{p_2/d_2}{\rightsquigarrow} H$ and $\tau' = G \overset{p_2/d_2'}{\rightsquigarrow} Y \overset{p_1/d_1'}{\rightsquigarrow} H$ are in swap-correspondence, the projections $\tau|_{O_1}$ and $\tau'|_{O_1}$ are in swap-correspondence, too. Moreover, since shift equivalence is defined as the least equivalence relation including the swap-correspondence that is closed under sequential composition, the functor $_|_{\mathscr{O}_1} : \mathbf{Trans}(\mathscr{O}) \to \mathbf{Trans}(\mathscr{O}_1)$ preserves shift-equivalence. Thus, the functor $_||_{\mathscr{O}_1}$ above is well defined. □

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

The proposition above implies that two transition sequences in $\mathcal{O}$ whose projections in $\mathcal{O}_1$ are observably non-equivalent, are indeed non-equivalent in $\mathcal{O}$. This fact is interesting for view-based testing. It means that certain test results for a view, such as conflicts and causal dependencies, carry over to the overall system.

In this section, we have developed what looks like a canonical generalization of the DPO notions of independence and shift-equivalence to graph transitions. However, as already discussed briefly after Definition 4.1, there is more than one way of generalizing, for example, the basic definition of independence. In the following section, we shall explore in depth an alternative approach based on an implementation of transitions by DPO sequences.

5. Implementation equivalence of graph transitions

In Section 3, Theorem 3.5 provides a way of implementing the effect of a faithful transition by means of a DPO step plus additional context transformations. Thus, if we provide enough rules for the transformation of the context, graph transitions can be implemented by DPO transformation sequences. This implementation, which extends to a mapping from transition sequences to transformation sequences, allows us to define a notion of *implementation equivalence* on transition sequences by identifying all sequences whose implementations belong to the same shift-equivalence class.

This alternative approach to concurrent semantics for graph transitions will be investigated in this section. In particular, we compare the notion of independence of transitions established in the previous section with one obtained through the implementation. It turns out that independence of transitions is actually more general than the independence of the corresponding implementations because the former does not take into account dependencies between unspecified effects.

We extend a graph transformation system by a *universal context*, that is, a set of context rules modelling all 'allowed' interactions with the environment of the system. The idea is to introduce for each open type $t \in O^-$ a rule $-_t$ that deletes one item of that type, and for each open type $t \in O^+$ a rule $+_t$ creating one item of that type.

Definition 5.1 (Extension by context rules). Assume an open graph transformation system $\mathcal{O} = \langle TG, P, \pi, O \rangle$ with $O = \langle O^-, O^+ \rangle$. Then, the extension of $\mathcal{O}$ by the universal context for O is the graph transformation system $C(\mathcal{O}) = \langle TG, P', \pi' \rangle$ where

$$P' = P \cup \{-_t | t \in O^-\} \cup \{+_t | t \in O^+\} \text{ and } \pi'(p) = \pi(p) \text{ for all } p \in P \text{ with}$$
$$\pi'(-_t) = (G^t \hookleftarrow G^t_- \hookrightarrow G^t_-) \text{ and } \pi'(+_t) = (G^t_- \hookleftarrow G^t_- \hookrightarrow G^t).$$

Here G^t denotes the TG-typed graph generated by a single vertex or edge of type t, and G^t_- the subgraph of G^t with that vertex or edge removed (see, for instance, rules ρ^+ and ρ^- in Figure 17).

Now, transitions in $\mathcal{O}$ are implemented by transformation sequences in $C(\mathcal{O})$ based on the characterization of transitions as transformations plus change-of-context in Theorem 3.5.

Definition 5.2 (Implementation of transitions). Let $\mathcal{O} = \langle TG, P, \pi, \langle O^-, O^+ \rangle \rangle$, and $C(\mathcal{O})$ be given as above. Assume a transition $G \overset{p/d}{\rightsquigarrow} H$ in $\mathcal{O}$ and let $G' \overset{p/d'}{\Longrightarrow} H'$ be the DPO

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

transformation step with injections $e_G : G' \to G$ and $e_H : H' \to H$, which exists by Theorem 3.5 (see also Figure 12 on the right). An *implementation of* $G \overset{p/d}{\rightsquigarrow} H$ *in* $C(\mathscr{O})$ is a transformation sequence $\rho = \rho^-; p/d'; \rho^+ : G \to H$ in $C(\mathscr{O})$ where:

(i) In $\rho^- : G \to G'$ only context rules $-_t$ for $t \in O^-$ are applied, such that $e_G : G' \to G$ is the constructed injection.
(ii) In $\rho^+ : H' \to H$ only context rules $+_t$ for $t \in O^+$ are applied, such that the constructed injection is $e_H : H' \to H$.

Moreover, given a sequence $\tau = p_1/d_1; p_2/d_2; \ldots$ of transitions in $\mathscr{O}$, an implementation of τ in $C(\mathscr{O})$ is a transformation sequence $\sigma = \rho_1; \rho_2; \ldots$ in $C(\mathscr{O})$ where ρ_i is an implementation of p_i/d_i for $i = 1, \ldots, n$.

It follows from Theorem 3.5 that for each transition sequence in $\mathscr{O}$ there exists an implementation by a transformation sequence σ in $C(\mathscr{O})$.

Although the implementation of transition sequences is not unique, it becomes functorial if we replace concrete transformation sequences by shift-equivalence classes. Also, this functor is surjective, that is, each shift-equivalence class in $C(\mathscr{O})$ implements a transition sequence in $\mathscr{O}$ as the following proposition shows.

Proposition 5.3 (Surjectivity of implementation). Let $\mathscr{O} = \langle TG, P, \pi, \langle O^-, O^+ \rangle \rangle$, and $C(\mathscr{O})$ be given as above. Then, the implementation relation above defines a surjective functor $impl : \mathbf{Trans}(\mathscr{O}) \to \mathbf{Trafo}(C(\mathscr{O}))/_{sh}$ which is the identity on objects and where $impl(\tau) = [\sigma]_{sh}$ if σ implements τ.

Proof. It is easy to see that, given a transition $G \overset{p/d}{\rightsquigarrow} H$ as above, all its implementations ρ are shift equivalent. In fact, the only freedom we have is the scheduling of the atomic rules in ρ^- and ρ^+, where, however, the resulting bottom spans are fixed in advance. It follows that all implementations of a transition sequence τ are equivalent since shift equivalence is closed under sequential composition. Thus, the functor *impl* is well defined.

Moreover, this functor is surjective, that is, each equivalence class $[\sigma]_{sh}$ in $C(\mathscr{O})$ contains a transformation sequence σ' that implements a transition sequence in $\mathscr{O}$. Consider a transformation sequence ρ and a decomposition $\rho_1; \rho_2; \ldots$ such that $\rho_i^-; p_i/d_i; \rho_i^+$ for $i = 1, \ldots, n$ as in Definition 5.2. It is easy to verify such a decomposition of ρ exists since the ρ_i^- or ρ_i^+, or both, may be empty. It follows from Theorem 3.5 that each ρ_i implements a faithful transition in $\mathscr{G}$. The construction of $C(\mathscr{O})$ ensures that this transition satisfies the frame conditions O. This implies by Definition 5.2 that σ is an implementation of the corresponding transition sequence. □

Next, we define an equivalence for transition sequences based on the implementation functor and the notion of shift equivalence for transformation sequences.

Definition 5.4 (Implementation equivalence on transition sequences). We assume that $\mathscr{O} = \langle TG, P, \pi, \langle O^-, O^+ \rangle \rangle$ is an open graph transformation system. Two transition sequences τ and τ' are *implementation equivalent* if their implementations $impl(\tau)$ and $impl(\tau')$ are shift equivalent DPO sequences. The *category of* impl-*concurrent transition*

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

sequences in $\mathcal{O}$ is the quotient $\mathbf{Trans}(\mathcal{O})/_{impl}$ of $\mathbf{Trans}(\mathcal{O})$ with respect to implementation equivalence.

Notice that the functoriality of the implementation, together with the fact that shift equivalence on transformation sequences is closed under sequential composition, entails that the implementation equivalence on transition sequences is also closed under sequential composition. Moreover, since implementation equivalence does not identify objects, the quotient $\mathbf{Trans}(\mathcal{O})/_{impl}$is well defined. Thus, we obtain the category $\mathbf{Trans}(\mathcal{O})/_{impl}$ whose objects are the same objects of $\mathbf{Trans}(\mathcal{O})$ and equivalence classes of transition sequences $[\tau]_{impl}$ as arrows.

It follows that the implementation functor is in fact an isomorphism between the category of impl-equivalence classes of transition sequences and the category of shift-equivalence classes of DPO sequences.

Corollary 5.5. The implementation functor $impl : \mathbf{Trans}(\mathcal{O}) \to \mathbf{Trafo}(C(\mathcal{O}))/_{sh}$ induces an isomorphism of categories $I : \mathbf{Trans}(\mathcal{O})/_{impl} \to \mathbf{Trafo}(C(\mathcal{O}))/_{sh}$ by $I([\tau]_{impl}) = [impl(\tau)]_{sh}$.

Proof. By Proposition 5.3, the implementation functor is a surjective functor between transition sequences and concurrent transformation sequences. Moreover, since two transition sequences are implementation equivalent if and only if their implementations are shift equivalent, the functor $I : \mathbf{Trans}(\mathcal{O})/_{impl} \to \mathbf{Trafo}(C(\mathcal{O}))/_{sh}$ defined by $I([\tau]_{impl}) = [impl(\tau)]_{sh}$ is an injective and a surjective functor. Hence, it is an isomorphism between the two categories. □

Let us relate this notion of concurrency to the one based on shift-equivalence of transitions as established in the previous section. In Example 4.5 we have discussed two kinds of concurrency, that is, *intra-concurrency* within the subsystem under consideration and *inter-concurrency* between the subsystem and its environment. In addition to intra- and inter-concurrency, we may now speak of *extra-concurrency* if we consider two operations of the environment. It is interesting to note a correspondence with three different ways of defining sequential independence of transitions. The most liberal definition of sequential independence would be the set-theoretic one based on the idea that matches and comatches intersect only in items that are preserved by both transitions. This means that we only take into account the dependencies between explicitly specified effects, thus capturing the idea of intra-concurrency.

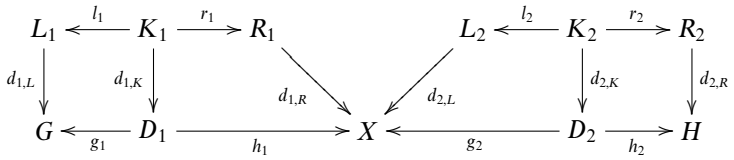
Definition 4.1, in addition, considers the relation between specified and unspecified effects, that is, it captures both intra- and inter-concurrency, but not extra-concurrency. This is only observed by the independence of the implementations, which also compares the unspecified effects of the first and second transitions. Thus, this latter notion of independence is the most restrictive one.

In order to make this more precise, we now compare the sequential independence of transitions (Definition 4.1) with the independence of the DPO sequences (Definition 2.11) obtained through implementation. As anticipated, sequential independence is reflected but not preserved by the implementation functor. The following proposition shows that the implementation functor reflects sequential independence.

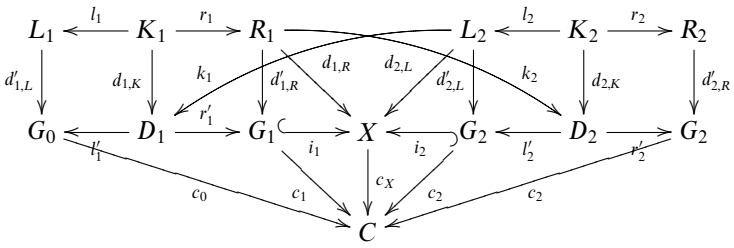
https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

Proposition 5.6 (Implementation reflects independence). Let $\tau = \tau_1;\tau_2$ be a transition sequence such that $impl(\tau_1)$ and $impl(\tau_2)$ are sequentially independent. Then, τ is sequentially independent also.

Proof. Let $\tau = \tau_1;\tau_2$ be a transition sequence as below with $\tau_i = p_i/d_i$



such that $impl(\tau_1) =: \sigma_1$ and $impl(\tau_2) =: \sigma_2$ are sequentially independent. Let us consider the DPO transformation steps p_1/d'_1 and p_2/d'_2, which are in σ_1 and σ_2, respectively. Since, by hypothesis, $\sigma_1;\sigma_2$ is sequentially independent, Definition 2.11 entails that there exist graph morphisms $k_1 : L_2 \to D_1$ and $k_2 : R_1 \to D_2$ such that $c_1 \circ r'_1 \circ k_1 = c_2 \circ d'_{2,L}$ and $c_2 \circ l'_2 \circ k_2 = c_1 \circ d'_{1,R}$ (see diagram below). Let G_1, (respectively, G_2) together with the corresponding morphisms, be the pushout graph of the morphisms r_1 and $d_{1,K}$ (respectively, l_2 and $d_{2,K}$). By Theorem 3.5, G_1 as well as G_2 are subgraphs of X. Using i_1 and i_2 to denote the corresponding inclusions, we have that $h_1 = i_1 \circ r'_1$ and $g_2 = i_2 \circ l'_2$. Finally, the nature of context rules entails that, $\langle c_i, C\rangle$ being the colimit of the DPO diagrams in $impl(\tau)$, we also have that $c_X \circ i_1 = c_1$ and $c_X \circ i_2 = c_2$. Therefore, we obtain the following commutative diagram:



From this we have that

$$c_X \circ h_1 \circ k_1 = c_X \circ i_1 \circ r'_1 \circ k_1 = c_1 \circ r'_1 \circ k_1 = c_2 \circ d'_{2,L} = c_X \circ i_2 \circ d'_{2,L} = c_X \circ d_{2,L}$$

and, analogously, $c_X \circ g_2 \circ k_2 = c_X \circ d_{1,R}$. Now, since the horizontal morphisms are injective morphisms, c_X is an injective morphism also, and, therefore, we have that $h_1 \circ k_1 = d_{2,L}$ and $g_2 \circ k_2 = d_{1,R}$, which proves that τ is sequentially independent. □

Finally, in the counterexample below, we show that the implementation functor does not preserve sequential independence since independence of transitions does not take into account the interference between unspecified effects, while they are considered in the implementation.

Example 5.7 (Implementation does not preserve independence). We consider a transition sequence τ as in Figure 16. It is clear that τ is sequentially independent, while its implementation is not sequentially independent. Consider the sequence of two double pushout transformations in the middle in Figure 17. They are clearly not sequentially

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

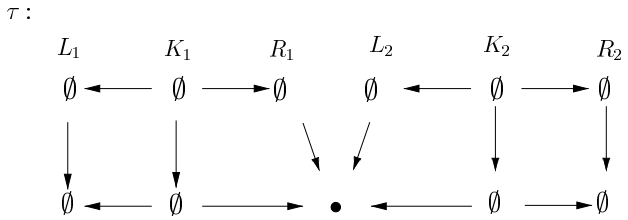


Fig. 16. sequential independence not preserved by implementation

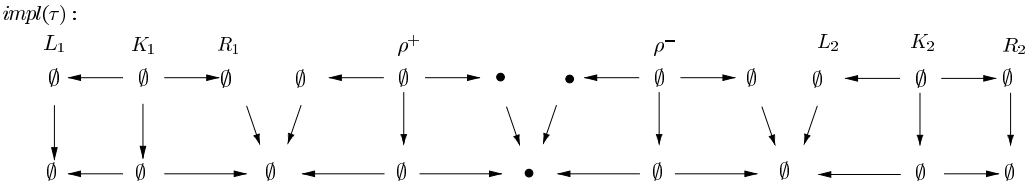


Fig. 17. sequential independence not preserved by implementation

independent since there is no morphism from a graph consisting of one node to the empty graph.

As a consequence, we have two different notions of independence and concurrency for graph transitions. The first, which is defined in Section 4 directly on transitions, generalizes the classical DPO notions in the obvious way. The second is obtained by lifting the DPO notions via the implementation functor.

6. Conclusion

In this paper we have developed the basics of a theory of independence and equivalence for double-pullback graph transitions. Based on two different generalizations of the corresponding notions for the DPO approach, two different variants are proposed and related to each other. The first is a straightforward application of the DPO definitions to DPB diagrams, and is called *shift equivalence* in analogy with the DPO case. The second is based on an implementation of transitions in terms of DPO transformations, and is therefore called *implementation equivalence*.

It turns out, that the implementation equivalence leads to a more restrictive notion of independence (that is, more causal dependencies and conflicts) because it takes into account dependencies between external operations. Instead, the notion of independence related to the shift equivalence only detects dependencies between internal operations, or between internal and external operations. These three kinds of concurrency resulting from our view-based approach are called, respectively, *intra-, inter- and extra-concurrency*.

The implementation requires a restriction to *faithful* transitions, which form the natural and stable counterpart of DPO transformation steps. This view is also supported by the results for the second equivalence. Based on a definition of independence analogous to the DPO case, a local Church–Rosser theorem for faithful graph transitions is shown and

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

a corresponding notion of shift-equivalence is defined. This concurrent semantics of DPB transitions is shown to be compatible with view relations, that is, the projection of states and behaviour induced by a view of a graph transformation system defines a functor between the corresponding categories of concurrent computations. This means that we can trust our observation of causal dependencies, which are indeed preserved under the embedding of a view into a bigger system.

The practical relevance of the theory developed in this paper is motivated by an example of a view-based model of a distributed information system, which draws an analogy between concepts of database systems like schema and instance, query and update operation, views, and synchronization of concurrency, and corresponding notions in the theory of DPO and DPB graph transformation.

References

Andries, M. and Engels, G. (1996). A hybrid query language for the extended entity relationship model. *Journal of Visual Languages and Computing* **7** (3) 321–352. (Special Issue on Visual Query Systems.)

Baldan, P., Corradini, A., Ehrig, H., Löwe, M., Montanari, U. and Rossi, F. (1999). Concurrent semantics of algebraic graph transformation. In: Ehrig, H., Kreowski, H.-J., Montanari, U. and Rozenberg, G. (eds.) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution*, World Scientific 107–188.

Bauderon, M. (1996) A category-theoretical approach to vertex replacement: The generation of infinite graphs. In: 5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94. *Springer-Verlag Lecture Notes in Computer Science* **1073** 27–37.

Bauderon, M. and Jacquet, H. (2001) Pullback as a generic graph rewriting mechanism. *Applied Categorical Structures* **9** (1).

Bernstein, P., Hadzilacos, V. and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*, Addison-Wesley.

Corradini, A., Montanari, U. and Rossi, F. (1996) Graph processes. *Fundamenta Informaticae* **26** (3,4) 241–266.

Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R. and Löwe, M. (1997) Algebraic approaches to graph transformation, Part I: Basic concepts and double pushout approach. In: Rozenberg, G. (ed.) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, World Scientific 163–245.

Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (eds.) (1999a) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, World Scientific.

Ehrig, H., Heckel, R., Llabres, M. and Orejas, F. (1999b) Basic properties of double pullback graph transitions. Technical Report 99-02, TU Berlin.

Ehrig, H., Heckel, R., Llabres, M., Orejas, F., Padberg, J. and Rozenberg, G. (2000) Double-pullback graph transitions: A rule-based framework with incomplete information. In: Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (eds.) Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98), Paderborn, November 1998. *Springer-Verlag Lecture Notes in Computer Science* **1764**.

Ehrig, H., Pfender, M. and Schneider, H. (1973) Graph grammars: an algebraic approach. In: *14th Annual IEEE Symposium on Switching and Automata Theory* 167–180.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press

Engelfriet, J. and Rozenberg, G. (1997) Node replacement graph grammars. In: Rozenberg, G. (ed.) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, World Scientific 1–94.

Heckel, R. (1998) *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems*, Ph. D. thesis, TU Berlin.

Heckel, R., Corradini, A., Ehrig, H. and Löwe, M. (1996) Horizontal and vertical structuring of typed graph transformation systems. *Mathematical Structures in Computer Science* **6** (6) 613–648.

Heckel, R., Ehrig, H., Wolter, U. and Corradini, A. (2001) Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. *Applied Categorical Structures* **9** (1).

Heckel, R., Engels, G., Ehrig, H. and Taentzer, G. (1999) A view-based approach to system modelling based on open graph transformation systems. In: Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (eds.) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, World Scientific.

Kreowski, H.-J. (1977) *Manipulation von Graphmanipulationen*, Ph. D. thesis, Technical University of Berlin, Dept. of Comp. Sci.

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag (Volume 5 of *Graduate Texts in Mathematics*.)

Object Management Group (1999) UML specification version 1.3. `http://www.omg.org`.

Reisig, W. and Goltz, U. (1983) The nonsequential behaviour of Petri nets. *Information and Control* **57** (2,3).

Rozenberg, G. (ed.) (1997) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, World Scientific.

Schürr, A., Winter, A. and Zündorf, A. (1999) The PROGRES approach: Language and environment. In: Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (eds.) *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, World Scientific 487–550.

Winskel, G. (1986) Event structures. In: Advances in Petri Nets 86–Part II. *Springer-Verlag Lecture Notes in Computer Science* **255** 325–392.

https://doi.org/10.1017/S0960129501003553 Published online by Cambridge University Press