Noname manuscript No.
(will be inserted by the editor)

Interval-Based Temporal Functional Dependencies: Specification and Verification

Carlo Combi · Pietro Sala

Received: date / Accepted: date

Abstract In the temporal database literature, every fact stored in a database may be equipped with two temporal dimensions: the valid time, which describes the time when the fact is true in the modeled reality, and the transaction time, which describes the time when the fact is current in the database and can be retrieved. Temporal functional dependencies (TFDs) add valid time to classical functional dependencies (FDs) in order to express database integrity constraints over the flow of time. Currently, proposals dealing with TFDs adopt a point-based approach, where tuples hold at specific time points, to express integrity constraints such as "*for each month, the salary of an employee depends only on his role*". To the best of our knowledge, there are no proposals dealing with interval-based temporal functional dependencies (ITFDs), where the associated valid time is represented by an interval and there is the need of representing both point-based and interval-based data dependencies. In this paper, we propose ITFDs based on Allen's interval relations and discuss their expressive power with respect to other TFDs proposed in the literature: ITFDs allow us to express interval-based data dependencies, which cannot be expressed through the existing point-based TFDs. ITFDs allow one to express constraints such as "*employees starting to work the same day with the same role get the same salary*" or "*employees with a given role working on a project cannot start to work with the same role on another project that will end before the first one*". Furthermore, we propose new algorithms based on B-trees to efficiently verify the satisfaction of ITFDs in a temporal database. These algorithms guarantee that, starting from a relation satisfying a set of ITFDs, the updated relation still satisfies the given ITFDs.

C. Combi
Department of Computer Science,
University of Verona, Italy
E-mail: carlo.combi@univr.it

P. Sala
Department of Public Health and Community Medicine,
Department of Computer Science,
University of Verona, Italy
E-mail: pietro.sala@univr.it

Keywords Temporal Functional Dependencies · Temporal Databases · Interval-based Functional Dependencies · Compass Structures · B-trees · Allen's Relations

1 Introduction

Many computer applications, such as accounting systems, geographical, health, and multimedia information systems, process control systems, and reservation systems, require the ability to represent and manage data changing over time. *Temporal databases* allow one to describe the temporal evolution of information by associating one or more temporal dimensions with the stored data [17]. The fundamental temporal dimensions associated with any fact stored in a temporal database are *valid time*, which describes the time when the fact is true in the modeled reality, and *transaction time*, which describes the time when the fact is current in the database and can be retrieved.

The association of temporal concepts with their temporal dimension may be *point-based* and *interval-based*. When it is point-based, a temporal concept is true at each associated time point. On the other hand, when the association is interval-based, the given temporal concept is true only on the overall associated interval and its truth over single time points (or sub-intervals) of the given interval cannot be asserted [5]. In temporal databases, most research focused on such a point-based kind of association and on the related issues in querying and managing point-based temporal data [16, 19, 20]. Nevertheless, some limitations of point-based temporal databases have been highlighted and some proposals have been done, allowing one to deal with both interval-based and point-based facts [26]. A further taxonomy for temporal query languages distinguish between *sequenced* and *non-sequenced* semantics [3, 4]. Through sequenced semantics, a temporal database is viewed as a sequence of atemporal database states holding at specific time points. On the other hand, in a non-sequenced temporal query, the user considers temporal dimensions as regular attributes and has to explicitly include in the query the required temporal conditions.

Specifying integrity constraints (such as keys, foreign keys, cardinalities, and so on) is an important part of modeling data: in this regard, temporal integrity constraints are (possibly) dynamic constraints for temporal databases [37]. Temporal constraints are usually expressed through languages based on first-order logic [37]. From among the various temporal integrity constraints for temporal data [7, 12, 13, 28], in this paper we focus on a special kind of temporal integrity constraints, called *temporal functional dependencies* [36]. Temporal functional dependencies (TFDs) add a temporal dimension to classical functional dependencies (FDs) in order to deal with temporal data [2, 32, 34–36]. As a matter of fact, the temporal dimension mostly considered is *valid time*. As an example, while FDs model constraints such as "*employees with the same role get the same salary*", TFDs can represent constraints like "*for any given month, employees with the same role have the same salary, but their salary may change from one month to the next one*" [2, 35] or "*current salaries of employees uniquely depend on their current and previous roles*" [9, 32].

To the best of our knowledge, TFDs proposed in the literature rely on some kind of point-based semantics, possibly extended to consider a fixed point-based temporal grouping when different temporal granularities, i.e., time partitions, are

considered [2, 9, 36]. However, as already pointed out in the literature, sometimes an interval-based semantics of temporal data is needed to represent them in a meaningful way. For example, when modeling clinical data, therapies need to be represented as inherently interval-based as their effect is not merely related to single-drug administrations but derives from the overall drug-based, therapeutic treatment during the given interval. Such an interval-based semantics has to be considered even when we need to specify integrity constraints and, in particular, temporal functional dependencies. In the case of therapies, we could have the requirement, for example, that a therapy for a patient can be extended only by the same physician who prescribed the original therapy. It is easy to see in this case that point-based temporal functional dependencies do not help, as the constraint does not restrict point-by-point the presence of therapies prescribed by different physicians but only constrains similar therapies having intervals that meet (i.e., the first one ends when the second interval starts).

According to the sketched scenario, in this paper we specifically focus on interval-based temporal constraints expressed through interval-based temporal functional dependencies (ITFDs): we extend and complete a first proposal dealing with interval-based temporal functional dependencies [11]. More precisely, we propose new ITFDs based on Allen's interval relations [1] and analyze their expressiveness by means of some simple examples extracted from a clinical domain, showing that ITFDs are able to capture some dependencies that cannot be represented through point-based TFDs, which have been recently considered within a unifying framework [9]. ITFDs may be used to express non point-based temporal constraints such as "*employees starting to work the same day with the same role in the same project get the same salary*" or "*employees with a given role working on a project cannot start to work with the same role on another project that will end before the first one*". As a completely new contribution, we then propose several algorithms for verifying whether a temporal database satisfies a given ITFD: we focus on algorithms for incremental verification of ITFDs on a temporal database. Incremental verification consists of checking that, starting from a temporal database satisfying a given ITFD, the considered update results in a temporal database still satisfying the given ITFD. All the proposed algorithms are based on auxiliary data structures that are essentially B-trees. Moreover, we consider the issue of maintaining such B-trees when new tuples are added to the database.

The paper is organized as follows. In Sect. 2, we introduce the main concepts related to point- vs. interval-based data models and to sequenced vs. non-sequenced semantics in temporal queries. Then we discuss there some main issues in the area of temporal integrity constraints for temporal databases. Finally, we focus there on the specific kind of temporal integrity constraints we consider in this paper, namely that of temporal functional dependencies, and consider the main related contributions. In Sect. 3, we introduce an example based on a real-world scenario, the management of medical data, that will be useful through the following sections to give an idea of how TFDs and ITFDs work. In Sect. 4, we introduce syntax and semantics of Interval-based Temporal Functional Dependencies and analyze their expressiveness by discussing several examples that cannot be captured by point-based TFDs. Sect. 5 introduces and exemplifies new algorithms based on B-trees for the incremental verification of ITFDs and describes how such B-trees have to be updated according to updates of the temporal database. Finally, Sect. 6

provides some concluding remarks and discusses further extensions of the current work.

2 Related work

In this section, we first present various concepts that have relevance to a temporal semantics for temporal databases and to the broad topic of temporal integrity constraints. Subsequently, we move to the specific kind of temporal integrity constraints we consider in this paper, namely that of temporal functional dependencies.

2.1 Interval-based vs. point-based, and sequenced vs. non-sequenced semantics in databases

Both in the area of temporal databases and in AI, several approaches for temporal data modeling and for temporal reasoning have been proposed, where the emphasis is on how to associate temporal concepts (facts, objects, entities, and so on) with their temporal dimension. A first attempt to classify different proposals distinguishes between *point-based* and *interval-based* approaches. In a point-based approach, a temporal concept is considered true at each time point associated with it. In an interval-based approach a temporal concept is considered true only on the overall interval associated with it and nothing can be said in general about its truth over single time points (or sub-intervals) of the given interval [5]. Among the several proposals in AI dealing with either point-based or interval-based approaches, we mention here the work of Shoham, who tried, in some sense, to overcome the duality of the two approaches: Shoham proposes a first-order logic for dealing with the truth of propositions over intervals [24]. In particular, the author observes that the truth of a proposition over an interval is related to its truth over other intervals. The author classifies propositions depending on relations that have to be considered in order to determine their truth. Just to exemplify Shoham's approach, let us consider some kind of propositions. A proposition type x is *downward-hereditary* (written $\downarrow x$) if whenever it holds over an interval it holds over all of its sub-intervals, possibly excluding the two endpoints. For instance, "*John played less than forty minutes*" is downward-hereditary. Analogously by symmetry, a proposition type x is *upward-hereditary* (written $\uparrow x$) if whenever it holds over all the sub-intervals of a given interval, possibly excluding the two endpoints, it also holds over the given interval itself. For instance, "*The airplane flies at* 35000 *feet*" is upward-hereditary. A proposition type x is *solid* if it never holds over two properly overlapping intervals. For instance, the proposition "*The plane executed the LANDING procedure (from start to finish)*" is solid. As we will see in the following, some interesting properties that can be expressed using Shoham's proposition types cannot be captured by a point-based formalism.

Moving to the area of temporal databases, most approaches have been inherently point-based, and several research efforts have been devoted to querying and managing temporal databases according to a point-based semantics. As an example, if two or more tuples are timestamped through intervals but with a point-based semantics, they need to be *coalesced* to produce a single tuple if they

have the same values for all the atemporal attributes and overlapping valid time intervals, respectively [16, 19, 20]. Focusing on the expressiveness of temporal data models with an interval-based semantics, Terenziani and Snodgrass analyze the inadequacy of point-based semantics concerning models of natural language [26]. They propose a dichotomy between two types of fact. Facts are either *telic* (from the Greek "telos" meaning goal) or *atelic* (the Greek 'a' as a prefix indicates negation). Telic events are characterized by the fact that they reach a *culmination* (e.g., "*John won the lottery*"), while atelic facts do not have an intrinsic culmination (e.g., "*John is building a house*"). Moreover, Terenziani and Snodgrass [26] propose an algebraic framework which deals with combinations of telic and atelic facts and they show how to add these concepts to a temporal query language (SQL/Temporal [25]). Terenziani et al. show that current point-based database approaches have some limitations that do not allow the representation of an important class of temporal medical data (i.e., telic data) [27]. They propose a new three-sorted model and a query language that overcome such limitations by supporting both telic, atelic, and atemporal relations, and some coercion functions to move from telic to atelic interpretations and vice versa. In the context of multimedia temporal object databases, Combi [6] shows how to deal with different types of interval-based textual observations associated with a range of frames of a movie, by extending and adapting the proposition types introduced by Shoham [24] to distinguish different interval-based semantics for multimedia data.

A further distinction for temporal query language constructs is between *sequenced* and *non-sequenced* semantics [3, 4]. A temporal query with a sequenced semantics considers the temporal database as a sequence of atemporal database states, each of them holding at a given time point. On the other hand, a temporal query with a non-sequenced semantics considers temporal dimensions of a temporal database as regular attribute values without enforcing any temporal semantics. It is then responsibility of the user to possibly enforce some temporal semantics in the query by explicitly specifying conditions involving attributes representing temporal dimensions. The distinction of sequenced and non-sequenced semantics for temporal queries is intertwined with the distinction between point-based and interval-based temporal data in a nontrivial way. In general we could say that the sequenced semantics reflects in a strict sense the point-based semantics of temporal data. This is because temporal data are considered point by point in a sequence of states and only data holding at the same state are considered for join, selection, grouping, and so on. On the other hand, non-sequenced semantics considers temporal queries on both point-based and interval-based temporal data, as the temporal meaning of data is completely and explicitly managed by the user.

2.2 Temporal integrity constraints

Integrity constraints, both temporal and atemporal, are an important part of a database schema; indeed, they express properties that have to be satisfied by any database (instance) of that schema. A database satisfying all the integrity constraints is called *consistent*. Integrity constraints are commonly expressed in a declarative way using logic. Usually, such integrity constraints do not provide any hint on how to keep the database consistent when data are inserted, deleted and modified. Thus, as we will see in the following for our proposal, it is extremely

important to propose efficient procedures for checking and enforcing such constraints. Integrity constraints are thus relevant at different levels and in various database-related tasks: from the conceptual and logical design, to reasoning on data, to the support of efficient data storage and retrieval [29].

Temporal integrity constraints express integrity constraints that can be dynamic (i.e., constraining data holding at different time points/intervals) and are checked over temporal databases [37]. They allow the user to represent constraints such as "*a patient who had an adverse reaction to a given drug, cannot receive that drug later on*". Languages for expressing such temporal constraints are usually based on first-order logic with explicit timestamps or with temporal modal operators [37]. Certain classes of temporal constraints (for specific data models) have been explicitly considered for their practical importance and general applicability. From among them, we mention here temporal integrity constraints for conceptual data models [7, 13, 28], temporal integrity constraints for XML data [12], and temporal functional dependencies [36]. As for temporal integrity constraints in conceptual modeling, one of the issues considered is related to the representation of temporal constraints in temporally-extended Entity-Relationship (ER) models [28]. For example, Combi et al. propose the temporal conceptual data model TimeER, supporting the specification of advanced temporal constraints like temporal keys, time-invariant keys, and temporal superclass/subclass relationships [7]. TimeER allows one to represent constraints such as "*the SSN of a person cannot be reassigned*" or "*a tracking code for a package identifies the given package, but it can be reused after the first package ends its validity*". Another aspect considered is how to extend existing conceptual models to express (even temporal) cardinality constraints. For example, Currim et al. analyse spatio-temporal semantics for cardinality constraints by introducing the concept of evaluation window to express constraints such as "*over the course of a month, an employee may choose to participate in no more than 3 projects*" [13]. Recently, the problem of temporal constraints in semi-structured and XML data has been considered [10, 12]: Combi et al. propose a graph-based generic model able to uniformly represent semi-structured data and their temporal aspects [10]. In particular, they consider in a formal and systematic way both valid and transaction times, together with the set of temporal constraints needed to correctly manage the semantics of the represented time dimension. The authors discuss operations which allow the incremental management of the proposed model satisfying the introduced temporal constraints. For example, it is possible to represent temporal constraints such as "*relation Organizes between a Person node and a Conference (child) node cannot be established before that Person node is valid in the considered domain*". Currim et al. consider the case for XML documents where past versions of documents are retained [12]. The authors describe how to interpret temporal constraints both as sequenced constraints, applicable at each point in time, and as non-sequenced constraints, i.e., across time. Different types of constraints are considered (e.g., key constraints, cardinality constraints). As an example, it is possible to express (non-sequenced) constraints such as "*there are between 0 and 4 supplier URLs in the temporal document over a period of any calendar month*".

2.3 Temporal functional dependencies

We now move closer to the main kind of temporal constraints considered in this paper. Some classes of temporal constraints, expressible through a restricted syntax, have been specifically studied because of their (even practical) importance. In particular, several kinds of temporal functional dependencies (TFDs) have been proposed in the literature, usually as temporal extensions of the widely known (atemporal) functional dependencies [36]. A (atemporal) functional dependency (FD) over a set of attributes U is an expression $X \rightarrow Y$ where $X, Y \subseteq U$. A relation r over U satisfies the FD $X \rightarrow Y$ if for all tuples $t_1, t_2 \in r$, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

In the following, we provide a short overview of the main formalisms for TFDs proposed in the literature. Jensen et al. propose a bitemporal data model that allows one to associate both valid and transaction times with data [18]. They define TFDs as FDs that must be satisfied at any bitemporal point (i.e., representing both valid and transaction times: *chronon* in the authors' terminology). More formally, TFDs are defined as follows [18]: Let X and Y be sets of non-timestamp attributes of a temporal relation schema $R^B = R(U|T)$. A database instance r^B of R^B satisfies a TFD $X \rightarrow^T Y$ iff for each bitemporal time point the FD $X \rightarrow Y$ holds, i.e., $\forall t_1, t_2 \in r(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$, where r is the atemporal relation containing all tuples of r^B holding (valid and current) at the considered bitemporal time point. As an example, let *Emp* be a temporal relation schema with the set of atemporal attributes $U = \{empId, salary, role, project\}$. The condition "*at any time, the salary of an employee uniquely depends on his role*" can be expressed by TFD $role \rightarrow^T salary$.

Bettini, Jajodia, and Wang's notion of TFD takes advantage of time granularity [33]: time granularity is a partition of a time domain in groups of indivisible units called *granules*. Examples of granularities are Day, Month, and WorkingDay. A time granularity is associated with each relation schema. Informally, a TFD $X \rightarrow_H Y$ is satisfied by a temporal relation associated with a granularity G if and only if for all tuples t_1, t_2, if (i) $t_1[X] = t_2[X]$, (ii) t_1 and t_2 hold at time granules of G contained in a single granule of H, then $t_1[Y] = t_2[Y]$. Bettini, Jajodia, and Wang's TFDs allow one to specify conditions on tuples associated with granules of a given granularity and grouped according to a coarser granularity. As an example, if we consider the temporal relation schema *Emp* with attributes $\{empId, salary, role, project\}$ and associated with granularity Month, the constraint "*for any given year, employees with the same role cannot have different salaries the same year; however, their salary may change from one year to the next one*" is captured by TFD $role \rightarrow_{\mathsf{Year}} salary$.

A general formalism for TFDs on complex (temporal) objects has been proposed by Wijsen [35]. It is based on a data model that extends the relational model with the notion of object identity, which is preserved through updates, and with the ability of dealing with complex objects, that is, objects that may have other objects as components. The meaning of TFD $c : X \rightarrow_\alpha Y$ can be intuitively explained as follows: Let t_1 and t_2 be two objects of class c at time points i and j, respectively, where (i, j) belongs to the time relation α, which is a binary relation on the time domain. If t_1 and t_2 agree on X, then they must agree on Y as well. It has been shown that the class of Wijsen's TFDs subsumes the class of Bettini et al.'s TFDs [35]. More precisely, Bettini et al.'s TFDs are exactly the TFDs on

chronologies (i.e., time relations representing granularities). In earlier work, Wijsen introduces a special notation for some relevant subclasses of TDFs [34]. In particular, he abbreviates $X \rightarrow_{Next} Y$ as X **N** Y and for $X \rightarrow_{Forever} Y$ as X **G** Y. Wijsen's TFDs allow one to specify conditions on tuples grouped according to any given time relation. As an example, it is possible to express the condition "*employees cannot have different salaries over two consecutive time points if their role does not change*" by means of TFD $Emp : empId, role$ **N** $salary$.

Vianu proposes a simple extension to the relational model in order to describe the evolution of a database over time [32]. According to it, a temporal database is viewed as a sequence of instances (states) over time. A change in the state of the database is produced by the execution of an update, an insertion or a deletion. A *database sequence* is a sequence of consecutive instances of the database, together with "*update mappings*" from one instance (the "old" one) to the next instance (the "new" one). Tuples are viewed as representations of domain objects. Since a tuple and its updated version represent the same object, tuples preserve their identity through updates. According to Vianu's notation, for each attribute A, $\overset{\vee}{A}$ represents its old value and $\hat{A}$ its new value. For each set U of attributes, let $\overset{\vee}{U} = \{\overset{\vee}{A} \mid A \in U\}$ and $\hat{U} = \{\hat{A} \mid A \in U\}$. Constraints on the evolution of attribute values of tuples (objects) over time are expressed by means of dynamic functional dependencies (DFDs): A DFD over U is an FD $X \rightarrow Y$ over $\overset{\vee}{U}\hat{U}$ such that, for each $A \in Y$, both $XA \cap \overset{\vee}{U} \neq \emptyset$ and $XA \cap \hat{U} \neq \emptyset$. The above condition on DFDs ensures that $X \rightarrow Y$ does not imply any nontrivial FD over $\overset{\vee}{U}$ or $\hat{U}$. As an example, the condition: "*new salaries of employees depend uniquely on their current and previous roles*" is captured by the DFD $\overset{\vee}{role}\ \overset{\wedge}{role} \rightarrow \overset{\wedge}{salary}$ over the set of attributes $U = \{empId, salary, role\}$.

Combi et al. propose a framework for TFDs that subsumes and extends the above proposals [9]. This framework uses a simple temporal relational data model based on the notion of *temporal relation*, i.e., a relation extended with a timestamping temporal attribute VT.

Two temporal views, respectively called *next* and *nexttuple*, have been introduced. They allow one to join tuples that satisfy a specific temporal relation in order to represent relevant cases of (temporal) evolution. Such views may be considered as a more powerful relational counterpart of the "*update mappings*" introduced by Vianu. For example, given a temporal distance k, with $k \geq 1$, the view *next* allows one to join pairs of corresponding tuples at temporal distance k. More precisely, given a temporal relation schema R, with attributes $U \cup \{VT\}$, a temporal relation r on R, and a pair of tuples $t, t' \in r$, the application of the view *next* to r, denoted $\chi_Z^{r,k}$, with $Z \subseteq U$ and $k \geq 1$, joins t, t' if (and only if) $t[Z] = t'[Z]$ and $t'[VT] = t[VT] + k$. The schema of the resulting relation is $ZW\overline{W} \cup \{VT, \overline{VT}\}$, where $W = U - Z$.

With the introduced data model, and leveraging temporal views, TFDs may be expressed by the syntax $[E\text{-}Exp(R), t\text{-}Group]X \rightarrow Y$, where $E\text{-}Exp(R)$ is a relational expression on R, called *evolution expression*, $t\text{-}Group$ is a mapping $\mathbb{N} \rightarrow 2^{\mathbb{N}}$, called *temporal grouping*, and $X \rightarrow Y$ is a functional dependency.

As for the semantics, similarly to the case of standard FDs, a TFD is a statement about admissible temporal relations on a temporal relation schema R. A temporal relation r on the temporal relation schema R satisfies a TFD

$[E\text{-}Exp(R), t\text{-}Group]X \to Y$ if it is not possible that the relation obtained from r by applying the expression $E\text{-}Exp(R)$ features two tuples t_1, t_2 such that:

(i) $t_1[X] = t_2[X]$,
(ii) $t_1[VT]$ and $t_2[VT]$ (the same for $t_1[\overline{VT}]$ and $t_2[\overline{VT}]$, if present) belong to the same temporal group, according to the mapping $t\text{-}Group$, and
(iii) $t_1[Y] \neq t_2[Y]$.

In other words, FD $X \to Y$ must be satisfied by each relation obtained from the evolution relation by selecting those tuples whose valid times belong to the same temporal group.

Consider, for example, the requirement "*for any given year, employees with the same role cannot have different salaries the same year; however, their salary may change from one year to the next one*". We previously represented this using a TFD of Bettini, Jajodia, and Wang. Within the proposed framework, it can be represented as $[Emp, \mathsf{Year(i)}]role \to salary$. The constraint "*new salaries of employees depend uniquely on their current and previous roles*", previously represented by a DFD a la Vianu over the set of attributes $U = \{empId, salary, role\}$, is now represented by the TFD[1] $[\chi^{Emp}_{empId}, \mathsf{Top(i)}]\overline{role}, role \to \overline{salary}$. It is also possible to express some TFDs having both a grouping a la Wijsen and a dynamic constraint a la Vianu. For example, the requirement "*year by year, salaries of employees depend uniquely on their current and previous roles*" is formalized as $[\chi^{Emp}_{empId}, \mathsf{Year(i)}]role, \overline{role} \to \overline{salary}$. Further constraints such as "*every year, employees with the same role, who will not change it from the current month to the next one, will get the same (unchanged) salary*" may be expressed. Existing TFD systems propose alternative extensions to the relational model, often introducing non-relational features (this is the case with Wijsen's objects [35] and Vianu's update mappings [32]), thus making it difficult to precisely evaluate their relative strength and their limitations. These differences complicate identifying the frameworks' distinctive features and to systematically compare them. Combi et al. [9] show how their proposed point-based framework subsumes the other TFDs previously described in the literature. In the following, we establish that interval-based temporal functional dependencies extend the possible temporal constraints we may specify for a given database with respect to the unifying framework proposed by Combi et al. for point-based temporal functional dependencies.

3 A motivating example

We now consider a simple clinical example for point-based TFDs described in Sect. 2 and then discuss some constraints that cannot be expressed with point-based TFDs.

Most health care institutions collect a large quantity of clinical information about patient and physician actions, such as therapies and surgeries, as well as about health care processes, such as admissions, discharges, and exam requests. All these pieces of information are temporal in nature and the associated temporal dimension needs to be carefully considered in order to be able to properly represent

[1] The simplified notation χ^r_Z instead of $\chi^{r,k}_Z$ is used in the following, when $k = 1$.

#	TherType	PatId	Phys	DrugCode	Qty	B	E
1	antiviral	1	Dorian	0458	300	1	16
2	analgesics	1	Cox	0976	200	2	10
3	cardiovascular	1	Turk	0118	100	3	8
4	antipyretics	1	Cox	0976	100	9	11
5	sedative	1	Turk	0345	10	13	15
6	anxiolytic	1	Dorian	0345	10	17	19
7	antiviral	2	Kelso	0458	200	1	10
8	cardiovascular	2	Quinlan	0118	100	4	7
9	analgesics	2	Reid	0976	150	5	9
10	antiviral	2	Reid	0458	300	8	14
11	antiviral	1	Dorian	0789	200	1	18

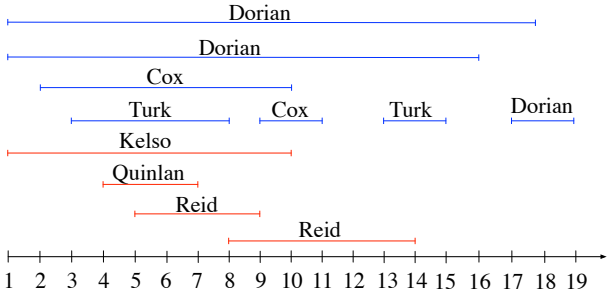

Fig. 1 An instance of relation *PatTherapies*, storing data about patient therapies and its representation on the time line with values for attribute *Phys*

clinical data and to reason about them [8]. In this section, we briefly introduce a real-world example taken from clinical medicine, namely that of patient therapies.

Suppose we have patients who undergo several different therapies: each therapy can be supervised by a physician, and consists of the administration of some drug to the patient. Information about patients and therapies is stored in a relation according to the schema $PatTherapies(TherType, PatId, DrugCode, Qty, Phys, B, E)$, where *TherType* identifies a type of pharmacological therapy, *PatId* represents a patient ID, *DrugCode* and *Qty* the prescribed drug and its quantity, respectively, and *Phys* the physician who made the prescription (and is responsible for the therapy). Finally, attributes B and E represent the beginning and end time points of the tuple valid interval, respectively: they represent the bounds of the interval specified by the physician for each therapy. An instance of *PatTherapies* is provided in Fig. 1.

As an example, for the given relation schema, the requirement *"at any time, the quantity of the prescribed drug depends on the drug and on the type of therapy"* can be expressed by Jensen's TFD $TherType, DrugCode \rightarrow^{T} Qty$, while the requirement "*every month, the physician who prescribes a given drug depends on the therapy*" may be captured by Bettini, Jajodia, and Wang's TFD $TherType \rightarrow_{\mathsf{Month}} Phys$. Moreover, it is possible to express by Wijsen's TFD $PatTherapies : PatId, DrugCode \rightarrow_{16days} Qty$ the requirement *"for every patient, the quantity of a drug cannot change within 16 days"*: this means that we have to wait 16 days without prescribing a drug to a patient if we want to change the quantity of this drug for that patient. It is worth noticing that this kind of constraint cannot be expressed by Bettini, Jajodia, and Wang's TFDs since no granule can overlap another one. Finally, the requirement: *"the new quantity for a drug depends only on the old quantity"* may be captured by Vianu's DFD

$\overset{\vee}{DrugCode}, \overset{\vee}{Qty} \rightarrow \overset{\wedge}{Qty}$. As we already noticed, all these TFDs may be expressed in a homogeneous way in the framework recently proposed by Combi et al. [9]. Moreover, new TFDs may be expressed in the proposed framework. For example, the TFD $[\chi^{PatTherapies}_{PatId,TherType}, \mathsf{Month(i)}]PatId, Qty \rightarrow \overline{Qty}$ specifies that therapies of the same type for a given patient administered in consecutive time points require that the drug quantity of the second administration depends only on the drug quantity of the first administration.

Let us now move to further, more complex constraints and suppose that our database has to satisfy the following new ones:

Example 1 Some policies of the hospital may be described as in the following:

1. Every patient may receive several therapies at the same time from different physicians, but overlapping therapies for the same patient must be prescribed by the same physician. In other words, if a patient during a therapy needs another therapy which lasts beyond the end of the current therapy, then this therapy must be prescribed by the same physician who prescribed the other one;
2. Every day, there is a single individual responsible (i.e., a physician) for all the ending therapies with the same drug;
3. A patient cannot start on the same day two therapies of the same type with different drugs;
4. Therapies of a given drug for a patient must have the same quantity when the period of a therapy immediately follows after another one for the same drug. In other words, if a patient's therapy with a given drug starts when the previous therapy with the same drug ends, the administered drug quantities must be the same.

It is easy to see that in order to verify these policies through the acquired data, both the start points and the end points of every pair of tuples come into play. Thus, the point-based TFDs proposed in Sect. 2.3 cannot be used to specify the above requirements related to the hospital policy, as they do not allow one to distinguish start/end points when facts begin/finish to hold from time points when facts (continue to) hold. Moreover, we might be interested in expressing even for interval-based tuples some constraints such as, for example, that a patient cannot have two different therapies with the same drug on the same day. This constraint is inherently point-based so we need a way to represent it even when tuples are interval timestamped.

4 Interval-based functional dependencies

In this section we first recall interval relations and the related notation we will use throughout the paper. Then we introduce the temporal relational data model we adopt. Finally, we propose a new type of temporal functional dependency based on Allen's interval relations and provide some meaningful examples.

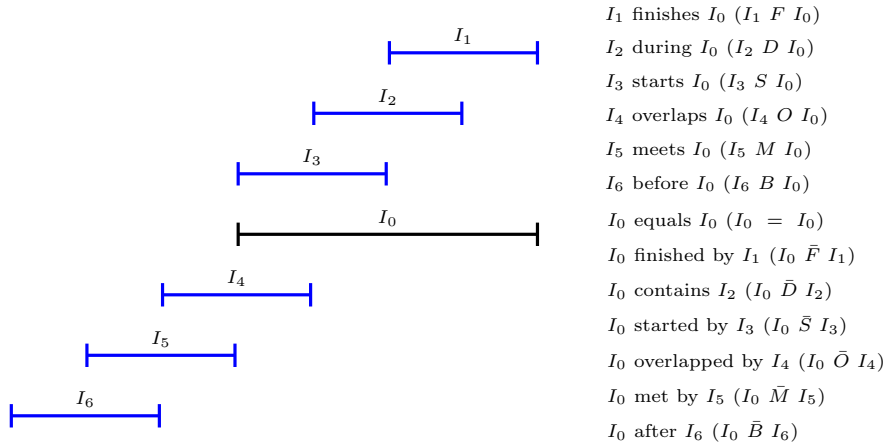

Fig. 2 The thirteen Allen relations between intervals

4.1 Interval relations

Given a totally ordered set $\mathbb{O} = \langle O, \leq \rangle$, an interval I over $\mathbb{O}$ is a pair $I = [b, e]$ where $b, e \in O$ and $b \leq e$. For any interval $I = [b, e]$ over $\mathbb{O}$ let $points(I)$ denote the set of points in O between b and e: $points(I) = \{p \mid p \in O \text{ and } b \leq p \leq e\}$. While the possible distinct relations between two points considering only the linear order are reduced to three (equality, successor, and predecessor), considering the order among the two endpoints of two intervals leads us to have thirteen possible relations. These relations are depicted in Fig. 2 according to the notation proposed by Allen in [1]. It is worth noting that every relation has its dual obtained by switching the position of the two intervals. Consider, for example, two intervals $I_1 = [b_1, e_1]$ and $I_2 = [b_2, e_2]$: we have that I_1 D I_2 (I_1 *during* I_2), if and only if $b_2 < b_1 < e_1 < e_2$. By reverting the arguments, we have that I_2 $\overline{D}$ I_1 (I_2 *contains* I_1), if and only if $b_2 < b_1 < e_1 < e_2$, which is equivalent to I_1 D I_2. More precisely, given two intervals $I = [b, e]$ and $I' = [b', e']$ we say that:

(1) $I = I'$ iff $b = b'$ and $e = e'$;
(2) I M I' iff $e = b'$;
(3) I S I' iff $b = b'$ and $e < e'$;
(4) I F I' iff $b > b'$ and $e = e'$;
(5) I O I' iff $b < b'$ and $b' < e < e'$;
(6) I D I' iff $b' < b$ and $e < e'$;
(7) I B I' iff $e < b'$.

4.2 The interval-based temporal relational data model

In discussing our new functional dependencies based on intervals within a relational framework, we use a simple temporal (relational) data model based on the concept of temporal relation. A temporal relation r is a relation on a temporal relation schema $\mathcal{R}$ defined on attributes $U \cup \{B, E\}$, where U represents a set of atemporal

attributes and B, E are the temporal attributes describing the valid interval of a tuple. We assume that the domain of both attributes B and E is a totally ordered set $\mathbb{O}$. Clearly, a tuple $t \in r$ satisfies $t[B] \leq t[E]$. We recall that, assuming the underlying domain for attributes A_1 and A_2 has a total order, atomic formulas for comparing tuples are either of the form $t[A_1]\ \theta\ t'[A_2]$ or of the form $t[A_1]\ \theta\ c$, with $\theta \in \{=, \neq, <, \leq, >, \geq\}$, A_1, A_2 being attribute names, c a constant value and t, t' tuples of relation r. To avoid ambiguities in the terminology employed, in the following we will use *(temporal) instance* for "(temporal) relation" and will let *relation* refer to Allen's interval relations.

4.3 Interval-based temporal functional dependencies

Let us now consider the basic definition of an *Interval-based Temporal Functional Dependency* (ITFD). In the following, we will only deal with interval relations in the set $\mathcal{A} = \{S, F, B, M, D, O, =\}$. Indeed, in this case it is not meaningful to distinguish between a relation and its dual, as it will be clear from the following definition of interval-based temporal functional dependency.

Definition 1 Let X and Y be sets of atemporal attributes of a temporal relation schema $\mathcal{R} = R(U, B, E)$ and $\sim$ an Allen's interval relation. An instance r of $\mathcal{R}$ satisfies an ITFD $X \rightarrow_{\sim} Y$ if for each pair of tuples t_1 and t_2 such that $[t_1[B], t_1[E]] \sim [t_2[B], t_2[E]]$ and $t_1[X] = t_2[X]$, it is also true that $t_1[Y] = t_2[Y]$.

Basically, ITFDs group tuples whose B and E attribute values satisfy the interval relation $\sim$. In the above definition, all the possible tuples having as valid interval either $[b, e]$ or $[b', e']$, where $[b, e] \sim [b', e']$ are considered together. If there exist two tuples having their valid intervals related through the considered relation $\sim$, respectively, and both tuples agree on (the tuple of) values of atemporal attributes X, then the ITFD imposes that both tuples must agree on (the tuple of) values of atemporal attributes Y. In the following, we will use notation $X \rightarrow_{\{\sim^1, \sim^2, \ldots \sim^n\}} Y$ with $\sim^1, \sim^2, \ldots \sim^n \in \mathcal{A}$, as a shorthand for $\{X \rightarrow_{\sim^1} Y, X \rightarrow_{\sim^2} Y, \ldots X \rightarrow_{\sim^n} Y\}$.

As already mentioned, we focus only on (sub) set $\mathcal{A}$ of Allen's interval relations, without considering the dual ones. Indeed, dual relations are not needed for the specification and verification of ITFDs, because ITFDs are based on the equality of the considered (atemporal) values. Thus, each (ordered) pair of tuples satisfying an interval relation will satisfy also the dual one, where tuples will be considered in the pair with the opposite order. In other words, any ITFD with a given interval relation implies also the corresponding ITFD with the dual relation (and vice versa).

Let us now consider the first requirement expressed in Example 1 of Sect. 3: it can be rephrased as "*overlapping drug administrations for a given patient must have the same physician*". This constraint can be expressed by the ITFD

$$PatId \rightarrow_O Phys.$$

A time-oriented graphical account of tuples of relation *PatTherapies* is provided in the lower part of Fig. 1. As we may notice, the instance satisfies ITFD $PatId \rightarrow_O Phys$ only for tuples related to the patient with $PatId = 1$. Dr. Cox added a therapy *antipyretics*, but the related valid interval is contained in the

interval of therapy *antiviral* prescribed by Dr. Dorian. Tuples related to therapies of patient with $PatId = 2$ instead do not satisfy ITFD $PatId \rightarrow_O Phys$, as both intervals of therapies prescribed by Dr. Reid overlap a therapy prescribed by another physician. This kind of property cannot be expressed with point-based TFDs. Basically the lack of expressiveness depends on the fact that point-based TFDs refer only to database snapshots which are either evaluated in isolation or grouped together according to some granularity or joined to the next snapshot to consider some kind of tuple evolution. In our example, intervals of patient therapies are considered in a holistic way and the considered temporal dependency is not checked against all the database snapshots, as it has to consider the specified interval relation(s) and thus, in general, both start and end points of intervals. In our example, *Dorian* starts a therapy on the patient with $PatId = 1$, and *Cox* starts therapy *analgesics* for the same patient during this therapy. After that, *Cox* adds another therapy (*antipyretics*). Suppose that this last therapy of *Cox* would last until time 14: then, either this tuple or the tuple related to therapy by *Turk* having valid time $[13, 15]$ violates the ITFD. Moreover, suppose that tuples are inserted according to the start of their valid time: in this case, the insertion of tuple #5 would be blocked.

Let us consider the schema *PatTherapies* and suppose that we want to express the second constraint specified in Example 1 of Sect. 3 "*every day, there is a single responsible (i.e., physician) for all the ending therapies with the same drug*": such requirement is expressed by the ITFD

$$DrugCode \rightarrow_F Phys.$$

The following constraint "*a patient cannot start on a single day two therapies of the same type with different drugs*" may be expressed as

$$PatId, TherType \rightarrow_S DrugCode.$$

The last constraint in Example 1 "*(temporally) meeting therapies of a given drug for a patient must have the same quantity*" is expressed as

$$PatId, DrugCode \rightarrow_M Qty.$$

Finally, let us consider the point-based constraint considered in Sect. 3 "*a patient cannot have two different therapies with the same drug the same day*". It is easy to see that the ITFDs

$$PatId, DrugCode \rightarrow_{\{S,F,O,D,M,=\}} TherType, Qty, Phys$$

capture this property: they correspond to that of snapshot key (i.e., a key constraint on each database snapshot in isolation), similarly to the point-based functional dependencies by Jensen, Snodgrass, and Soo [18]. Indeed, all these ITFDs together prevent any instance of *PatTherapies* from having two tuples with intersecting valid times, the same corresponding attribute values for $PatId, DrugCode$ and different values for (even only one of) attributes $TherType, Qty, Phys$. Tuples with the same corresponding attribute values for $PatId, DrugCode$ and different values for (even only one of) attributes $TherType, Qty, Phys$ may exist in any instance only if their valid times occur one before the other (i.e., they are disjoint).

This property can be simply proved by observing that the set of interval relations $\{S, F, O, D, M, =\}$ considered by the above ITFDs covers all the possible cases of intersection between two given intervals. Indeed, any relation of this set requires that the two intervals have some common time points (at least one, as in the meet relation M) and only the relation *before* B is not considered by the given ITFDs, as it is the sole one requiring the considered intervals to be disjoint.

To conclude this section, let us consider the example in Sect. 2 about employees. It is represented in our interval-based context by the temporal schema $Emp(U, B, E)$, where $U = \{empId, salary, role, project\}$. The two constraints "*employees starting to work the same day with the same role in the same project get the same salary*" and "*employees with a given role working in a project cannot start to work with the same role on another project that will end before the first one*" previously introduced may be expressed, respectively, by the two ITFDs below:

$$project, role \rightarrow_S salary \qquad empId, role \rightarrow_D project.$$

5 Efficiently verifying the satisfaction of ITFDs

In this section we present a set of algorithms (and discuss their complexity) for checking the satisfaction of a given ITFD $X \rightarrow_\sim Y$ over a temporal instance r with schema $R(U, B, E)$ where $X, Y \subseteq U$. In the following, we mainly consider the (sub) set $\{S, F, B, M, D, O\}$ of Allen's interval relations without considering the equality relation. Indeed, the equality relation does not appear to be of particular interest since the expression of ITFDs with the equality relation may be reduced to consider point-based TFDs on start and end time points.

Verifying the satisfaction of $X \rightarrow_\sim Y$ may be considered in two different but intertwined ways: i) given an instance r of R, check whether or not r satisfies $X \rightarrow_\sim Y$, ii) given an instance r of R satisfying $X \rightarrow_\sim Y$ and a tuple t, verify whether $r \cup \{t\}$ still satisfies $X \rightarrow_\sim Y$. We call the first problem *checking ITFD satisfaction*, while the second one is called *incremental ITFD verification*. It is not difficult to see that these two problems are closely related. In fact, checking ITFD satisfaction reduces to the incremental ITFD verification by adopting the algorithm developed for this problem and, starting from $i = 0$ with instance $r_0 = \emptyset$ with schema R, incrementally verifying whether $r_i \cup \{t_i\}$ with $t_i \in r \setminus r_i$ satisfies ITFD $X \rightarrow_\sim Y$. If the update of r_i with t_i still verifies $X \rightarrow_\sim Y$, then $r_{i+1} = r_i \cup \{t_i\}$, $i = i + 1$ and the algorithm is applied again. If r satisfies $X \rightarrow_\sim Y$, after $|r|$ iterations we can determine ITFD satisfaction. Some complexity improvements to this naive approach can be done. This is because when checking ITFD satisfaction we know the whole instance r in advance, while in the update problem tuples are only given one after each other and must be immediately checked for $X \rightarrow_\sim Y$-satisfaction with the current r.

If $r \cup \{t\}$ satisfies the given ITFD, t is inserted and then our instance becomes $r' = r \cup \{t\}$. In the case of incremental verification of ITFDs, we have to take into account that all the auxiliary (i.e., indexing) data structures, possibly used to efficiently verify the ITFD satisfaction, need to be properly updated. According to this view, the overall problem of incremental ITFD verification has to deal

with two aspects: (i) efficiently verifying the ITFD satisfaction, and (ii) efficiently updating the indexing structures used for the verification.

The ideas behind algorithms for the incremental ITFD verification apply with few modifications to obtain algorithms for checking ITFD satisfaction. Therefore, we describe and discuss in detail algorithms for the incremental ITFD verification problem and then in some relevant cases we point out minor differences with algorithms for checking ITFD satisfaction.

In our algorithms proposed here, we adopt an approach commonly found in the database area: we consider the so-called *disk complexity*. Basically, we assume a main memory of size M and an input of size $T \gg M$. The input (i.e., the instance) is stored in the secondary storage system (a disk). As usual, we assume that each disk read/write operation is able to load in the main memory a large amount of contiguous bytes named *block*. Let P be the size of each block; then we have that the main memory can keep only $k = \lfloor M : P \rfloor$ blocks. Standard disk read/write operations have a cost (i.e., time) which is of the order of 10^5 higher than the cost of the corresponding main memory operation. Thus, in the incremental ITFD verification we consider how to minimize the number of disk operations and evaluate our algorithms with respect to disk complexity (i.e., the number of read/write disk accesses). According to this perspective, we consider ad-hoc, suitable *B-trees* as index structures [30]. B-trees are balanced tree data structures that keep data sorted and allow data access operations in logarithmic time. They are optimized for (database) systems that read and write large blocks of data, as each node of a B-tree corresponds to a block. The structure of any B-tree node may be represented as $\langle P_1, (k_1, v_1), P_2, (k_2, v_2), P_3, (k_3, v_3), \ldots, P_{q-1}, (k_{q-1}, v_{q-1}), P_q \rangle$, where P_i for $i = 1, 2, ...q$ stands for a pointer to a B-tree node (such pointers are null in leaf nodes), k_i, for $i = 1, 2, \ldots q-1$, stands for the key used for sorting data, while v_i for $i = 1, 2, \ldots q-1$ is either the data value associated with the corresponding key or a pointer to such data. Informally, the number $q-1$ of key-value pairs is managed to guarantee that nodes (except the root) are at least half-full. Within each node it holds $k_1 < k_2 < \ldots < k_{q-1}$, and for all search key values K in the subtree pointed at by P_i, it holds $k_{i-1} < K < k_i$ for $1 < i < q$, where $K < k_i$ for $i = 1$ and $k_{i-1} < K$ for $i = q$. In the following, we use such data structure to efficiently manage the incremental evaluation of different ITFDs. As we will see, keys and data values are suitably defined according to the considered ITFD[2].

In the following part of this section, we first define and describe a suitable spatial representation for interval-based tuples and briefly recall the definition and some useful properties of the standard data structures we use for the proposed algorithms. Then, for each interval relation $\sim \in \{S, F, B, M, D, O\}$, we describe the proposed algorithm for solving the incremental ITFD verification problem and discuss for the most important cases how to update the index structures used.

[2] In Fig. 9 and in the following ones, B-tree nodes are represented as rounded boxes, pointers are represented as small circles with pointing arrows, while key components are represented by suitable labels separated by commas, and value components are represented by suitable labels within round brackets.

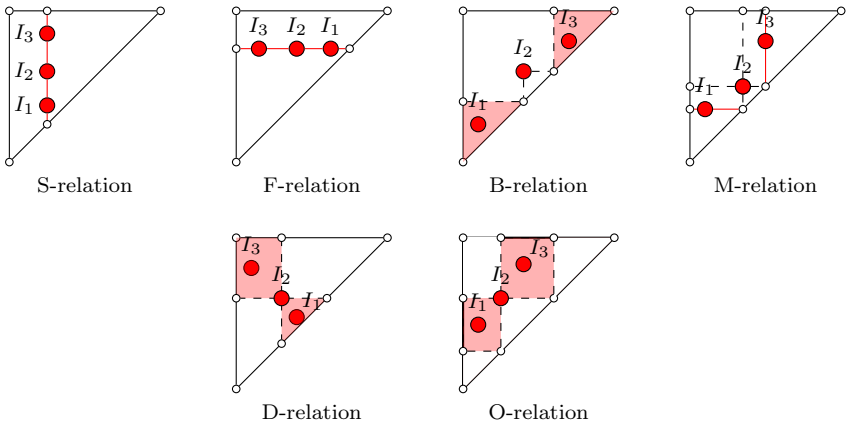

Fig. 3 Compass structures and translation of interval relations into relations between points: for each relation areas/lines are highlighted where points are in the considered relation (or its dual one) with point representing interval I_2. The figure represents only an initial and finite triangular region of the (infinite) octant

5.1 Representing sets of interval-based tuples

Let us start by proposing a geometric representation of intervals (called a *compass structure*) and the corresponding interpretation of interval relations. We will then use them in what follows to build indexing structures and algorithms for incremental ITFD verification.

Definition 2 Given a totally ordered set $\mathbb{D}$, the *interval set* $\mathbb{I}(\mathbb{D})$ is the set $\mathbb{I}(\mathbb{D}) = \{(x, y) \in \mathbb{D}^2 \mid x \leq y\}$.

Definition 3 Given a finite set of points $\mathbb{P} \subseteq \mathbb{I}(\mathbb{R})$ and a finite set of elements called colors $\mathcal{C} = \{c_1, \ldots, c_n\}$, a *compass structure* is a function $\mathcal{G}_{(\mathbb{P},\mathcal{C})} : \mathbb{P} \to \mathcal{C}$.

There exists a natural spatial representation of compass structures in the second "octant" of the Euclidean plane, i.e., the sector of the plane delimited by the y positive half-axis and by the bisector of the first quadrant. Indeed, as depicted in Fig. 3, any interval $[x, y]$ corresponds to a point (x, y) on the Euclidean plane and any such point (x, y) must be in the second octant of the plane as $x, y \geq 0$ and $x \leq y$. In the following figures, we represent the (infinite) octant as a (finite) triangle, i.e., its first part, moving up from the origin of axes. Intuitively, as we will detail in the following, the color associated with a time point in this representation corresponds to some relevant information. This representation is a variation of the original compass structure introduced by Venema [31] for axiomatizing and proving undecidability for a powerful interval temporal logic: recently, it has been successfully used to obtain decidability results for other interval temporal logics [22, 23].

There is a correspondence between positions of any two time points of the compass structure and the interval relation between the corresponding intervals. Fig. 3 gives a representation of each interval relation. Relations S and F are true

for intervals corresponding to points vertically and horizontally aligned, respectively. To understand whether an interval is before or after another interval I_2, we have to project its corresponding point (x, y) both horizontally and vertically on the bisector of the first quadrant, as depicted in Fig. 3. Indeed, its projections correspond to points (y, y) and (x, x), respectively, and help to delimit two triangular areas of the octant containing points representing intervals that are before and after I_2. For example, intervals corresponding to points I_1 and I_3 in Fig. 3 are before and after the interval corresponding to point I_2, respectively. The same approach may be applied to understand the other relations. For relation *meets* (M), the projection (x, x) of point (x, y), representing interval I_2, identifies the horizontal line of all points (having the same ordinate y) representing intervals finishing when I_2 starts. Fig. 3 depicts three points corresponding to intervals I_1, I_2, and I_3, where I_1 M I_2 and I_2 M I_3. The same projections are used when we consider relation D. Indeed, points (x, y), (x, x), and (y, y) identify the triangle containing all points corresponding to intervals that are during I_2, corresponding to (x, y). On the other hand, the rectangular region of the octant in the up-left position with respect to (x, y) contains all points corresponding to intervals containing I_2. Fig. 3 represents points corresponding to intervals I_1, I_2, and I_3, where I_1 D I_2 and I_2 D I_3. Finally, for relation *overlaps* O and the projection (x, x) for the point (x, y), we may identify the region contained between the horizontal line having y-coordinate x, the upper horizontal line having y-coordinate y, the y-axis, and the vertical line having x-coordinate x. This region contains all points corresponding to intervals overlapping interval I_2, represented by point (x, y). Similarly, we may identify the region of all points corresponding to intervals overlapped by interval I_2. Fig. 3 represents points corresponding to intervals I_1, I_2, and I_3, where I_1 O I_2 and I_2 O I_3.

Let us now consider how interval relations induce clusters in the compass structure. We start by defining relation $\rightarrow_\sim \subseteq \mathbb{P}\times\mathbb{P}$ as the relation $\{(p, p') \mid p \sim p' \vee p' \sim p\}$ and let $\rightarrow^*_\sim$ be its reflexive-transitive closure. We define the $\sim$-clusters as follows:

Definition 4 Given a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ and an interval relation $\sim \in \{S, F, B, M, D, O\}$, we say that a subset $\mathcal{C}l_\sim \subseteq \mathbb{P}$ is a *$\sim$-cluster* if and only if the following conditions hold:

1. (non-emptiness) $\mathcal{C}l_\sim \neq \emptyset$;
2. (transitive-closure) For every pair $p, p' \in \mathcal{C}l_\sim$ we have $p \rightarrow^*_\sim p'$;
3. (maximality) For every $p \in \mathbb{P}$, if there exists $p' \in \mathcal{C}l_\sim$ such that $p' \rightarrow^*_\sim p$, then $p \in \mathcal{C}l_\sim$.

It is easy to see that for any interval relation $\sim$, the set of all $\sim$-clusters induces a partition over $\mathbb{P}$.

Definition 5 Given a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, an interval relation $\sim \in \{S, F, B, M, D, O\}$, and a cluster $\mathcal{C}l_\sim$ of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, we say that $\mathcal{C}l_\sim$ is *consistent* if and only if for every pair $p, p' \in \mathcal{C}l_\sim$, we have $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(p) = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(p')$.

If a cluster $\mathcal{C}l_\sim$ is $\sim$-consistent, let $color(\mathcal{C}l_\sim)$ denote its color. Moreover, we say that a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ is $\sim$-consistent if all the $\sim$-clusters associated with $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ are $\sim$-consistent.

Let us now use compass structures to represent tuples involving two different sets of attributes X and Y. Given sets X, Y of attributes, we define a suitable compass structure for each tuple of values for X, representing all tuples of an instance having the given tuple of values for X. Colors of this compass structure are related to different (tuples of) values for set Y of attributes.

Definition 6 Given an instance r of a schema $R(U, B, E)$, two subsets $X, Y \subseteq U$, and a tuple v of values such that there exists $t \in r$ with $t[X] = \mathsf{v}$, we define *the compass translation of r on X with value v colored on Y* $\mathcal{T}r(r, X, \mathsf{v}, Y)$ as the compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with the following:

- $\mathbb{P} = \{(x, y) \in \mathbb{R}^2 \mid \exists t \in r(t[X] = \mathsf{v} \wedge t[B] = x \wedge t[E] = y)\}$,
- $\mathcal{C} = \{\mathsf{v}' \mid \exists t \in r(t[X] = \mathsf{v} \wedge t[Y] = \mathsf{v}')\}$,
- and for every $(x, y) \in \mathbb{P}$ we have $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = \mathsf{v}'$, if there exists $t \in r$ with $t[X] = \mathsf{v}$, $t[B] = x$, $t[E] = y$, and $t[Y] = \mathsf{v}'$.

Let us now consider ITFD $X \rightarrow_\sim Y$. We have a suitable compass structure for each tuple of values for attributes X: the associated compass structure has colors corresponding to different (tuples of) values for attributes Y. The verification that all tuples having the same values for X have also the same values for Y, when a given interval relation holds for these tuples, is reduced to verify that all clusters of all the related compass structures are consistent, i.e., correspond to a single color.
The following theorem links the $\sim$-consistency of a set of compass structures to the satisfaction of an ITFD $X \rightarrow_\sim Y$ on an instance r.

Theorem 1 *Given an instance r of schema $R(U, B, E)$ and two attribute sets X and Y with $X, Y \subseteq U$, ITFD $X \rightarrow_\sim Y$ holds for r if and only if for each v such that there exists $t \in r$ with $t[X] = \mathsf{v}$, $\mathcal{T}r(r, X, \mathsf{v}, Y)$ is $\sim$-consistent.*

Proof ($\Rightarrow$) Suppose by contradiction that there exists a value v for which there is an inconsistent cluster $\mathcal{C}l_\sim$ in the compass structure $\mathcal{T}r(r, X, \mathsf{v}, Y) = \mathcal{G}_{(\mathbb{P},\mathcal{C})}$. Then, there exists a pair of points (x, y) and (x', y') in $\mathcal{C}l_\sim$, for which $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) \neq \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y')$. By Definition 6, we have that there exists a pair of tuples t, t' in r with $t[X] = t'[X]$, $t[B] = x$, $t[E] = y$, $t'[B] = x'$, $t'[E] = y'$, and $t[Y] \neq t'[Y]$ (with $t[Y] = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y)$, $t'[Y] = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y')$). Moreover, we have by definition that either $[t[B], t[E]] \sim [t'[B], t'[E]]$ or $[t'[B], t'[E]] \sim [t[B], t[E]]$. Both cases violate $X \rightarrow_\sim Y$.
($\Leftarrow$) We reverse the proof for ($\Rightarrow$) and assume by contradiction that $X \rightarrow_\sim Y$ does not hold on r.

The last concepts we need to introduce are related to the identification of sets of intervals that can be only associated with tuples having specific values for attributes X and Y. For points representing intervals in compass structure, the following definition allows one to refer, for a given $\mathcal{C}l_\sim$, to the subsets of points p in $\mathbb{I}(\mathbb{R})$ that are "forced" to have color $color(\mathcal{C}l_\sim)$ if they eventually will be added to the compass structure.

Definition 7 Given a set of points $P \subseteq \mathbb{I}(\mathbb{R})$ and an interval relation $\sim \in \{S, F, B, M, D, O\}$, the closure of P according to relation $\sim$ is the set $closure_\sim(P) = \{p \in \mathbb{I}(\mathbb{R}) \mid \exists p' \in P(p' \rightarrow_\sim p)\}$.

Given a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ and a point $p' \notin \mathbb{P}$, we define the c'-extension of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with p' as the compass structure $\mathcal{G}_{\mathbb{P}\cup\{p'\},\mathcal{C}'}$ with

$$\mathcal{G}_{\mathbb{P}\cup\{p'\},\mathcal{C}'}(p) = \begin{cases} \mathcal{G}_{(\mathbb{P},\mathcal{C})}(p) & p \in \mathbb{P}; \\ c' & otherwise. \end{cases}$$

Given a $\sim$-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, a point p', and a color c', we say that $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ can be *consistently extended* if and only if the c'-extension of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with p' is a $\sim$-consistent compass structure. The following lemma points out which points cannot be inserted in a $\sim$-consistent compass structure.

Lemma 1 *Given a $\sim$-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ and clusters $\mathcal{C}l_\sim, \mathcal{C}l'_\sim$ in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with $color(\mathcal{C}l_\sim) \neq color(\mathcal{C}l'_\sim)$, for every c and every p, if the c-extension of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with p is $\sim$-consistent, then $p \notin closure_\sim(\mathcal{C}l_\sim) \cap closure_\sim(\mathcal{C}l'_\sim)$.*

Proof Let $\mathcal{G}_{\mathbb{P}\cup\{p\},\mathcal{C}'}$ be the c-extension of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with p for some color c and some point p. Suppose by contradiction that $\mathcal{G}_{\mathbb{P}\cup\{p\},\mathcal{C}'}$ is $\sim$-consistent and $p \in closure_\sim(\mathcal{C}l_\sim) \cap closure_\sim(\mathcal{C}l'_\sim)$. By Definition 5, we have that in $\mathcal{G}_{\mathbb{P}\cup\{p\},\mathcal{C}'}$ both $\{p\} \cup \mathcal{C}l_\sim$ and $\{p\} \cup \mathcal{C}l'_\sim$ are $\sim$-clusters, but they are not $\sim$-consistent because we would have $c = color(\mathcal{C}l_\sim)$ and $c = color(\mathcal{C}l'_\sim)$. By the hypothesis, however, we have $color(\mathcal{C}l_\sim) \neq color(\mathcal{C}l'_\sim)$ (contradiction).

This lemma constrains the position of a cluster with respect to the closure of other clusters. Basically it says that for every pair of different clusters $\mathcal{C}l_\sim, \mathcal{C}l'_\sim$ we have $\mathcal{C}l'_\sim \cap closure_\sim(\mathcal{C}l_\sim) = \mathcal{C}l_\sim \cap closure_\sim(\mathcal{C}l'_\sim) = \emptyset$. It is easy to observe that closures may intersect (and they usually do) and if colors of the respective clusters are different, then a point and a color do not exist that can be inserted in the intersection.

5.2 An overview of algorithms

In this section we provide a short example for each ITFD related to an interval relation. Moreover, we give an intuitive idea of how the corresponding compass structures are stored in a B-tree together with some informal operational behavior of both operations of insertion and deletion in presence of an ITFD $X \rightarrow_\sim Y$ with $\sim \in \{S, F, B, M, D, O\}$.

In the following examples, we assume that all the considered intervals are associated with tuples sharing the same corresponding values for attributes X. We also assume that a tuple t_j is visually represented by an interval I_j and its Y values $t[Y]$ are represented by the color (blue/red or dark/bright) of the interval; thus, we have for every i, j that $t_i[Y] = t_j[Y]$ if and only if I_i and I_j share the same color. Any picture associated with an example shows a sequence of operations on the instance; every snapshot of the current instance is enclosed in a box with rounded corners and the operation is described either above or below it. Tuples that are inserted (i.e., after their insertion, the instance still satisfies the considered ITFD) have the corresponding interval represented with a solid line. On the other hand, if the insertion of a tuple is forbidden, the interval associated with that tuple is represented by a dashed line.

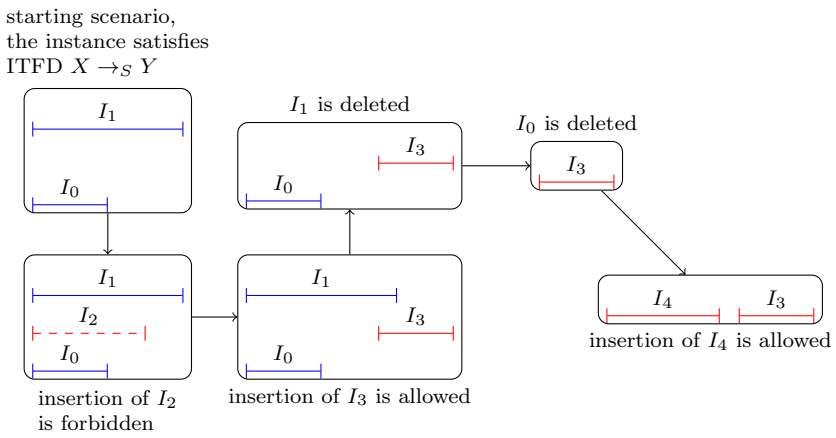

Fig. 4 An example of how tuple insertion is constrained when the instance has to satisfy ITFD $X \rightarrow_S Y$ (all the depicted tuples share the same corresponding values for attributes X and they have the same corresponding values for attributes Y if and only if they agree on the color of their intervals)

Checking $X \rightarrow_S Y$: Let us start with a simple instance r, depicted in Fig. 4, consisting of two tuples with associated intervals I_0 and I_1 and which satisfies an ITFD $X \rightarrow_S Y$. Both tuples agree on the value of attribute B (and of attributes X), so they must share the same value for attributes Y. If we try to insert tuple t_2, we have that I_2 S I_1 but t_2 has different values for attributes Y ($t_1[Y] \neq t_2[Y]$). Therefore, the insertion of tuple t_2 is forbidden. We then try to insert a tuple t_3 with $t_3[Y] \neq t_1[Y]$ but in a way that I_3 does not start and is not started by I_0: such insertion is allowed. After both tuples t_0 and t_1 are deleted, it is possible to insert tuple t_4 with associated interval I_4 with $t_4[Y] \neq t_1[Y]$. This is possible because both t_1 and t_2 have been deleted before the operation. We may conclude by observing that it suffices to keep a count of tuples starting at the same time point together with values for their Y attributes. Values for Y attributes must be the same for all these tuples. Thus, the corresponding compass structure is supposed to be S-consistent with respect to the ITFD $X \rightarrow_S Y$. For the incremental verification of ITFD $X \rightarrow_S Y$ on an instance r, we build a B-tree $\mathcal{BT}_S$ for every tuple of values c such that there exists $t \in r$ with $t[X] = c$. $\mathcal{BT}_S$ is indexed on the first coordinate $t[B]$ of tuple $t \in r$. For each such key $t[B]$, a B-tree node contains values $t[Y]$ and the number of tuples t' with $t'[Y] = t[Y]$ and $t[B] = t'[B]$. Since we build one B-tree for each tuple of values of attributes X in r, and we assume that r satisfies $X \rightarrow_S Y$, we can conclude that the tuple of values $t[Y]$ associated with any coordinate $t[B]$ inside a B-tree $\mathcal{BT}_S$ is unique. Every tuple t is checked against the B-tree $\mathcal{BT}_S$ associated with the tuple of values $t[X]$ using values $t[B]$ and $t[Y]$. If t can be inserted, the counter for the pair ($t[B]$, $t[Y]$) is incremented. The approach for checking ITFD $X \rightarrow_F Y$ is similar but uses $t[E]$ as the key value.

Checking $X \rightarrow_B Y$: Let us now consider the example depicted in Fig. 5. We start with an instance r, with two tuples t_0 and t_1, which satisfies ITFD $X \rightarrow_B Y$, since $t_0[Y] = t_1[Y]$ and I_0 B I_1. If we try to insert tuple t_2, we have that I_1 B I_2

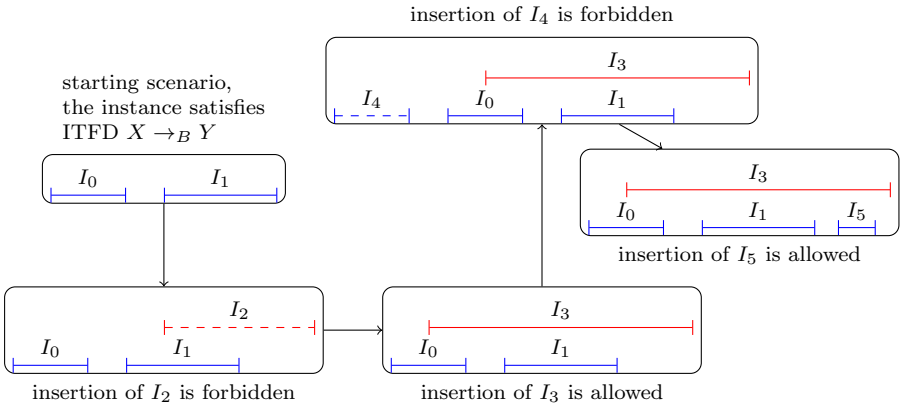

Fig. 5 An example of how tuple insertion is constrained when the instance has to satisfy ITFD $X \rightarrow_B Y$

and $t_1[Y] \neq t_2[Y]$. Then the insertion of t_2 into r is refused. It is worth noticing that a tuple t, for which $t[Y] \neq t_1[Y]$, may be inserted if and only if its associated interval intersects all the intervals associated with tuples having the same $t[X]$ in r, namely t_0 and t_1: this is so for tuple t_3. However, after the insertion of t_3 it is forbidden to insert any tuple t with associated interval I such that $I\ B\ I_0$ and $I\ B\ I_3$, because values for $t[Y]$ would be required to be equal to $t_0[Y]$ and $t_3[Y]$ at the same time (and this is impossible as $t_0[Y] \neq t_3[Y]$). That happens when we try to insert t_4, which shares the same values for Y attributes with tuples t_0 and t_1. The presence of tuple t_3 forbids the insertion of any tuple t having the associated interval before I_3, no matter what the values of $t[Y]$ are. Tuple t_5, on the other hand, may be inserted afterwards because $t_5[Y] = t_0[Y] = t_1[Y]$ and I_5 is after I_1 but not after I_3.

For the incremental verification of ITFD $X \rightarrow_B Y$, we make use of four B-trees $\mathcal{BT}_S$, $\mathcal{BT}_F$, $\mathcal{BT}_{I(S)}$, and $\mathcal{BT}_{I(F)}$ for each tuple of values which attributes X have in r. $\mathcal{BT}_S$ and $\mathcal{BT}_F$ are nearly the same B-trees used for ITFDs $X \rightarrow_S Y$ and $X \rightarrow_F Y$, respectively. They serve as auxiliary structures for efficiently updating $\mathcal{BT}_{I(S)}$ and $\mathcal{BT}_{I(F)}$, respectively. Let us focus on $\mathcal{BT}_{I(S)}$: its keys consist of ranges $[b, e]$ and its values are colours corresponding to some $t[Y]$. Each key $[b, e]$ is associated with the same color (corresponding to $t[Y]$) of all tuples starting within the given range, i.e., $b \leq t[B] \leq e$. This B-tree helps keep the verification time logarithmic, while $\mathcal{BT}_S$ is used to keep the time for updating $\mathcal{BT}_{I(S)}$ logarithmic. Any tuple t intended for insertion is verified against $\mathcal{BT}_{I(S)}$ by checking if there exist two keys $[b, e]$ and $[b', e']$ in $\mathcal{BT}_{I(S)}$ with different Y-values c and c' and with $t[E] < b < b'$ (the symmetric check is performed with $\mathcal{BT}_{I(F)}$ using $t[E]$ in place of $t[B]$ and vice versa). If such a pair of keys exists, then the insertion is refused. If there exist only one key $[b, e]$ with $t[E] < b$ and different Y values with respect to t, then the insertion is also refused.

Checking $X \rightarrow_M Y$: Let us consider instance r, consisting of two tuples t_0 and t_1, which satisfies ITFD $X \rightarrow_M Y$, as depicted in Fig. 6. If we try to insert tuple t_2 with $t_2[Y] \neq t_1[Y]$ and $I_0\ M\ I_2$, then the insertion of t_2 is forbidden. On the

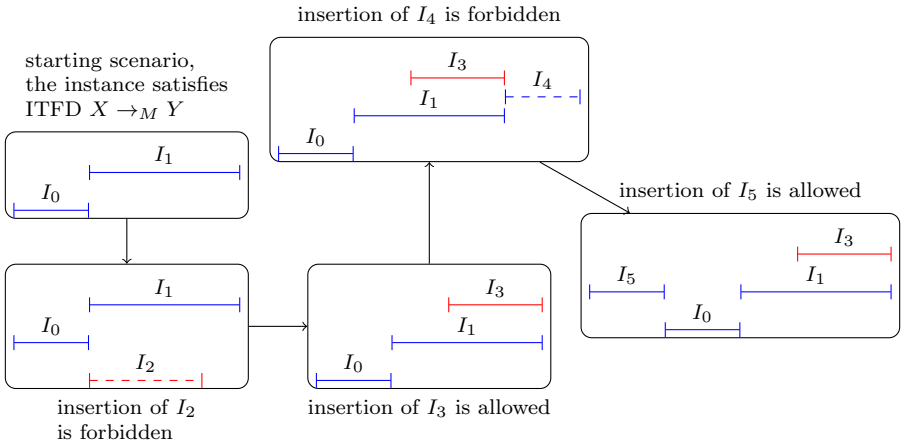

Fig. 6 An example of how tuple insertion is constrained when the instance has to satisfy ITFD $X \rightarrow_M Y$

contrary, tuple t_3, with $t_3[Y] \neq t_1[Y]$, is allowed for insertion in r since even if I_3 shares its right endpoint with I_1, it is not in relation *meets/met by* with I_1. In fact, we have I_3 F I_1. However, the new instance, which includes t_3, does not allow the insertion of any tuple t with an associated interval I which starts at the end time point of I_1 and I_3 ($t[B] = t_1[E] = t_3[E]$), no matter what the value of $t[Y]$ is (the new tuple should agree on Y values with both t_1 and t_2, which is impossible). This is so for tuple t_4, which is refused for insertion in the next step, while the insertion of tuple t_5 is allowed because $t_5[Y] = t_0[Y]$, I_5 M I_0, and there are no tuples t with associated interval I which are met by I_5 and having $t[Y] \neq t_5[Y]$.

For the incremental verification of ITFD $X \rightarrow_M Y$, we make use of one B-tree $\mathcal{BT}_M$ for each tuple of values of attributes X in r. The key of $\mathcal{BT}_M$ ranges over the set of all values $t[B]$ and $t[E]$, collected into a single set $\mathcal{S}$, for all tuples $t \in r$ having the same $t[X]$ associated with the given $\mathcal{BT}_M$. $\mathcal{BT}_M$ has a three-valued key:

- the first and most significant value is $v \in \mathcal{S}$,
- the second one is the orientation (it may be L for left or R for right, with $L < R$) that determines if the element is containing information about tuples that have $t[B] = v$ (left) or $t[E] = v$ (right),
- the third value is tuple c of attribute values for Y of tuple(s) ending/starting in v.

Finally, the value associated with a key consists of the number of tuples t with $t[Y] = c$ and $t[B] = v$ if the orientation is L ($t[E] = v$ if the orientation is R). Coordinate $t[B]$ of a tuple t considered for insertion is checked against $\mathcal{BT}_M$ by looking for a key $(t[B], R, c)$. If such a key exists and c is not equal to $t[Y]$, the insertion of tuple t is refused (a symmetric check is done for $t[E]$). On the other hand, if the tuple is accepted, at most two key-value pairs are inserted (in case of both $t[B] \notin \mathcal{S}$ and $t[E] \notin \mathcal{S}$) and the counters are updated accordingly.

Checking $X \rightarrow_D Y$: In the example shown in Fig. 7, we begin with an instance r consisting of four tuples (the instance satisfies ITFD $X \rightarrow_D Y$). The intervals associated with such tuples are arranged in order to form a D-cluster. It is

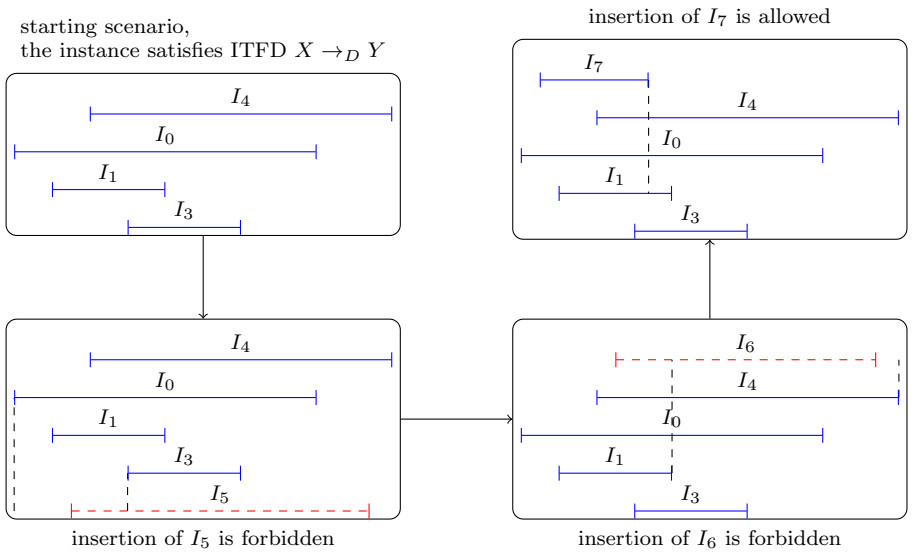

Fig. 7 An example of how tuple insertion is constrained when the instance has to satisfy ITFD $X \rightarrow_D Y$

worth observing that inside the cluster every tuple t_j represents either a maximum or a minimum for exactly one of the two attributes $t[B]$ and $t[E]$ that are the endpoints of interval I_j. In the case depicted in Fig. 7, we have that t_0 and t_3 represent the minimum and the maximum left endpoints, respectively, as relations $t_0[B] < t_1[B] < t_4[B] < t_3[B]$ hold. Also, t_1 and t_4 represent the minimum and the maximum right endpoints, respectively, as $t_1[E] < t_3[E] < t_0[E] < t_4[E]$ hold. For that particular arrangement of intervals inside the cluster it turns out that the insertion of tuple t_5, with $t_5[X] = t_0[X]$ and $t_5[Y] \neq t_0[Y]$, is refused since its left endpoint satisfies $t_0[B] < t_5[B] < t_3[B]$. By similar argument, it turns out that the insertion of tuple t_6, with $t_6[X] = t_0[X]$, $t_6[Y] \neq t_0[Y]$, is refused since $t_1[E] < t_6[E] < t_4[E]$. The last crucial insight highlighted by this example is that these two ranges $[t_0[B], t_3[B]]$ and $[t_1[E], t_4[E]]$ are sufficient to determine whether a tuple may be inserted in the cluster by checking when one of its endpoints happens to occur in the ranges considered, no matter how many tuples are contained in the cluster. Indeed, if we add a new tuple t_7 with $t_7[X] = t_0[X]$ and $t_7[Y] = t_0[Y]$, it turns out that t_7 is inserted and becomes part of the cluster. Moreover, since $t_7[E] < t_1[E] < t_3[E] < t_0[E] < t_4[E]$ hold, we have that t_7 substitutes t_1 as the representative of the minimum right endpoint. Hence, tuples that will be inserted afterwards will be checked against t_7 instead of t_1.

For the incremental verification of ITFD $X \rightarrow_D Y$, we exploit two properties of D-clusters (see Definition 4) now discussed. It is easy to see that in every D-consistent (we work under this assumption) compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, it is possible to give a total order $<$ relation over D-clusters. For every pair of D-clusters $\mathcal{C}l_1 \neq \mathcal{C}l_2$ in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, we are able to verify whether $\mathcal{C}l_1 < \mathcal{C}l_2$ or $\mathcal{C}l_2 < \mathcal{C}l_1$, considering at most four tuples per cluster (called cluster generators), without any concern

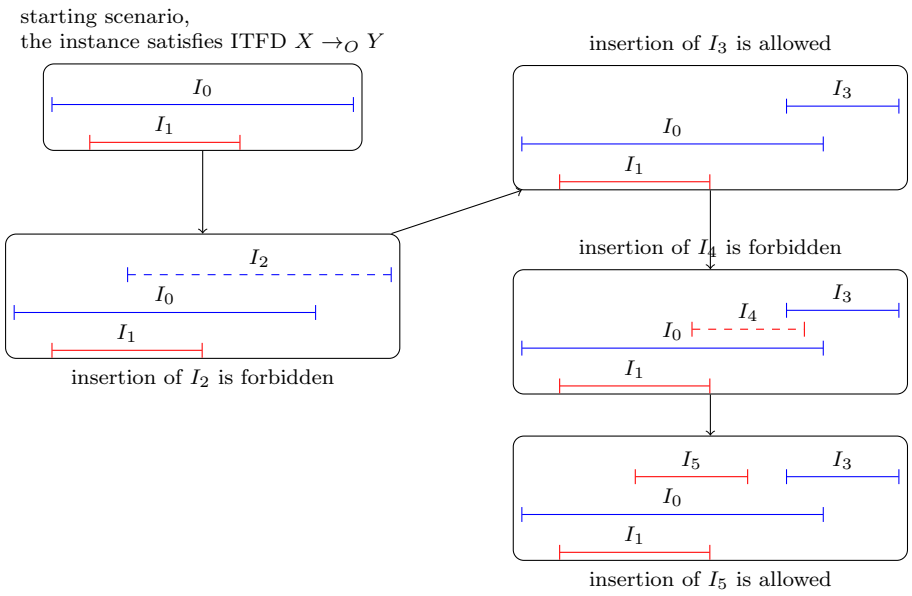

Fig. 8 An example of how tuple insertion is constrained when the instance has to satisfy ITFD $X \rightarrow_O Y$

for the number of tuples that a cluster contains. By restricting ourselves only to cluster generators in order to represent clusters, we can determine:

(i) the ordering relation among D-clusters,
(ii) whether a point is in or out a given D-cluster,
(iii) whether a point outside a given D-cluster belongs to a greater/lesser D-cluster or it originates a new D-cluster.

Thus, we build a B-tree $\mathcal{BT}_D$, where keys are the set of cluster generators and the value is the tuple of attribute values $t[Y]$ of tuples t in the cluster, which is unique under the assumption that $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ is D-consistent. Every tuple t to be inserted is checked against $\mathcal{BT}_D$ to verify if it belongs to a cluster: If so, when $t[Y]$ agrees with the value c of the cluster found, then t is inserted; otherwise it is refused. If t represents a new stand-alone cluster, then we do not need any further checking, and t is inserted in r together with a new key-value element in $\mathcal{BT}_D$ representing the D-cluster originated by t.

Checking $X \rightarrow_O Y$: Fig. 8 depicts instance r, which satisfies ITFD $X \rightarrow_O Y$. Tuples t_0 and t_1 with $t_0[X] = t_1[X]$ differ for the values of the Y attributes (i.e., $t_0[Y] \neq t_1[Y]$), but ITFD $X \rightarrow_O Y$ is still satisfied since I_0 does not overlap I_1 and vice versa. If tuple t_2 with $t_2[Y] = t_0[Y]$ is inserted and both I_0 O I_2 and I_1 O I_2 hold, then such an insertion is refused. A similar tuple t_3 with $t_3[Y] = t_0[Y]$ and I_0 O I_3, but neither overlapping nor overlapped by t_1, is allowed for insertion (and thus inserted) in r. Let us now consider tuple t_4 with $t_4[Y] = t_1[Y]$. If we plan to insert it at this point, we have that the insertion of t_4 would not be refused by the presence of t_0, as I_4 does not overlap and is not overlapped by I_0. However, it would

be refused by the presence of the newly inserted tuple t_3. Indeed, $t_3[Y] \neq t_4[Y]$ holds and the associated interval I_3 is overlapped by I_4. This suggests that a tuple t whose interval is overlapped by t_1 must have $t[Y] = t_1[Y]$ and $t[E] \leq t_3[B]$. That is so for t_5, which is successfully inserted since its associated interval I_5 does not overlap (and is not overlapped by) I_3.

For the incremental verification of ITFD $X \rightarrow_O Y$, we make use of two B-trees $\mathcal{BT}_H$ and $\mathcal{BT}_V$ for each tuple of values of X attributes in r. These two B-trees operate in a symmetrical way. When a new tuple t has to be inserted in r, B-tree $\mathcal{BT}_V$ allows us to efficiently check whether intervals (if any) that are overlapped by $[t[B], t[E]]$ share the same values for Y attributes. B-tree $\mathcal{BT}_H$ operates in a symmetric way to deal with intervals that possibly overlap $[t[B], t[E]]$. Such a dichotomy allows us to have a more efficient and simple representation for the O-consistent compass structures. For example, let us consider the case of B-tree $\mathcal{BT}_V$, which deals only with intervals that are overlapped by the interval associated with tuple t candidate for insertion. If we focus only on the overlapped-by relation, the compass structure is partitioned into vertical "stripes" identified by ranges $[lb, ub]$ over the set of endpoints of tuples in r (all tuples have the same attribute values for X associated with $\mathcal{BT}_V$). Each vertical stripe is represented in the B-tree $\mathcal{BT}_V$ by using its range $[lb, ub]$ as key (stripes are non-overlapping and so a trivial total order is given by the total order on their first endpoint). The corresponding value is composed by tuple c of values for Y attributes, and by two limit coordinates f and l (horizontal stripes are stored in $\mathcal{BT}_H$ in the very same way). Based on how relation O partitions the compass space, coordinate f identifies an upper limit for $t[E]$ of a (possibly inserted) tuple t with $lb \leq t[B] \leq ub$: if $t[E] \leq f$, any values are allowed for $t[Y]$. Coordinate l identifies a further limit for $t[E]$: a tuple t with $lb \leq t[B] \leq ub$ and $f < t[E] \leq l$ must have $t[Y]$ equal to c to be considered for insertion. Finally, the insertion of a tuple t such that $lb \leq t[B] \leq ub$ and $t[E] > l$ is refused (constraints provided by $\mathcal{BT}_H$ are symmetric). Intuitively, if the tuple t considered has $t[E]$ below or equal to f, there are no tuples overlapped by t and, thus, no constraints are given for $t[Y]$. If $t[E]$ is between f and l, t has to share the same tuple of values c for $t[Y]$ with the other tuples already in r and overlapped by t. If $t[E]$ is above l, then there are two groups of tuples with different values for Y overlapped by t and, thus, t cannot be considered for insertion as the given ITFD would be violated. A tuple t can be inserted in r without violating ITFD $X \rightarrow_O Y$ if and only if t, checked against both $\mathcal{BT}_V$ and $\mathcal{BT}_H$, does not generate any contradiction with respect to the constraints considered.

The correctness of the algorithms we will discuss in detail in the following sections is grounded on the fact that we

- adopt the widely known and sound B-trees as index structures (with the related search/update algorithms) [30],
- use a representation of relation instances based on compass structures, and
- for such structures we formally proved the equivalence between a $\sim$-consistent compass structure and the ITFD satisfaction for the corresponding instance.

In the following algorithms, B-trees are used for storing sets of tuples in their nodes. Usually, the key of the node represents some total linear order given over the tuples, depending on the interval relation $\sim$ in the considered ITFD $X \rightarrow_\sim Y$. For instance, we will see in the case of *starts* interval relation how such key is the attribute B. Partitioning r into sets that share the same value for attribute B

ITFD	tuple insertion	tuple deletion	ITFD satisfaction checking
$X \rightarrow_S Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$
$X \rightarrow_F Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$
$X \rightarrow_B Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$
$X \rightarrow_M Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$
$X \rightarrow_D Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\|)$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$
$X \rightarrow_O Y$	$\mathcal{O}(\log(\|r\|))$	$\mathcal{O}(\|r\|)$	$\mathcal{O}(\|r\| \cdot \log(\|r\|))$

Table 1 The complexities for the tuple insertion, deletion, and ITFD satisfaction checking, by the proposed index-based incremental verification of ITFDs

provides us a natural total order for the set of such partitions, which is the linear order on the values of B. The value stored in the node is the pair represented by the value $t[Y]$ for some $t \in r$, which is unique since all the tuples that share the same value for the attributes X and B must share the same value for Y to satisfy ITFD $X \rightarrow_S Y$. Moreover the value of the node also stores the number of tuples that share the same value for the attribute B to deal efficiently with the deletion operation. In some other cases, we use additional B-trees for performing fast search on such representations of grouped and totally ordered sets of tuples. Let us, for example, consider the case of the *during* interval relation: we have that tuples are grouped into clusters, which may be seen as monochromatic (all the points inside a cluster have the same color) hourglass-shaped regions in the compass representation of intervals. Such clusters are totally linearly ordered and each cluster can be uniquely identified by two points in the compass structure and by its color. Therefore, the linear order among clusters may be stored in a B-tree with key-value pairs where the key consists of the two points that identify the cluster and the value is the color of all points of the cluster. At runtime, the insertion of a tuple may cause consecutive clusters to be merged into a single one. In such a case, the considered tuple belongs to multiple consecutive clusters and we have to verify that the color of each of them agrees with the color of the tuple. Such an operation is expensive and may lead to a linear complexity in the size $|r|$. We take advantage of the main property that the clusters to consider are consecutive in the order. To keep the insertion time logarithmic, we make use of an additional B-tree where each node contains the first and the last element of a maximal sequence of consecutive clusters which share the same color. A tuple is allowed for insertion if it belongs to at most one of such sequences; otherwise, its insertion would merge clusters with different colors. Once we have verified that the tuple can be inserted, we have to modify the two B-trees and, in the case the tuple causes the merging of some cluster, it may happen that we have to merge a linear number of nodes in the B-tree for clusters (in the B-tree for sequences of clusters at most one node is modified or inserted). We exploit again the fact that affected clusters are consecutive and thus in the B-tree we can tailor a subtree that represents such a set of clusters which leads to a merging operation in logarithmic time. The fact that we consider only consecutive nodes in a B-tree is intensively used also in the case of relation *overlaps* where we slightly modify the B-tree structure by adding labels on some edges of the B-tree to represent a constraint that holds on all the nodes of the subtree rooted in the target node of the edge.

Let us conclude this section by overviewing algorithmic complexity for ITFD incremental verification. As we already noted, we will have a forest of B-trees, one

for each distinct tuple of values for attributes X (each B-tree in its turn could be referenced by an overall indexing structure on X values). The introduced B-trees are organized in order to guarantee that both checking whether or not $r \cup \{t\}$ satisfies $X \rightarrow_\sim Y$ and (hopefully) updating B-trees to represent $r \cup \{t\}$ will be executed in logarithmic time with respect to $|r|$. These two operations correspond to *tuple insertion*, for which the complexity is logarithmic in $|r|$ for every ITFD $X \rightarrow_\sim Y$ with $\sim \in \{S, F, B, M, D, O\}$, as summarized in Table 1. The other operation on instances is *tuple deletion*[3]. Such an operation may appear simpler than tuple insertion by observing that if r satisfies $X \rightarrow_\sim Y$ then $r \setminus \{t\}$ for every $t \in r$ still satisfies $X \rightarrow_\sim Y$. Some complications may arise in updating the suitable B-trees owing to the particular data structures adopted for making tuple insertion logarithmic. Considering Table 1, we can easily notice that ITFDs $X \rightarrow_D Y$ and $X \rightarrow_O Y$ present a linear worst-case complexity for tuple deletion. Therefore, removing a tuple t may affect a linear portion of the B-tree storing the compass structure for ITFD $X \rightarrow_\sim Y$ and attribute values $t[X]$. It is easy to observe that the consistent maintenance of data structures allowing faster tuple insertion produces slower tuple deletion, when the overall instance must satisfy ITFD $X \rightarrow_\sim Y$. We plan to deal with these complexity issues in the future. Here, we focus on keeping tuple insertion logarithmic.

The incremental ITFD verification may be used also for checking ITFD satisfaction of the full relation instance. To this end, one can trivially build an algorithm to check whether an instance r satisfies some ITFD $X \rightarrow_\sim Y$ by simply starting with an empty relation r' and incrementally performing tuple insertion in r' for every tuple $t \in r$. If there exists a tuple t which is refused, then we can conclude that r does not satisfy $X \rightarrow_\sim Y$. Otherwise, if the procedure terminates with $r' = r$, we can conclude that r satisfies $X \rightarrow_\sim Y$. Since this algorithm iterates $|r|$ times tuple insertion, we can conclude that its complexity is $\mathcal{O}(|r| \cdot \log(|r|))$ for every ITFD $X \rightarrow_\sim Y$ with $\sim \in \{S, F, B, M, D, O\}$, as summarized in Table 1.

5.3 Verifying satisfaction of $X \rightarrow_S Y$ and $X \rightarrow_F Y$

Verifying ITFD $X \rightarrow_S Y$ is straightforward because of the particular shape of the corresponding compass structures. In fact, given a S-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, the closures of clusters in it represent vertical lines. Thus, it suffices to keep track, for every cluster of points (x, y) in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, of the starting point x and its color $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y)$ (both of them are common by definition to all the points of the cluster), and the number of points corresponding to intervals starting at x. Fig. 9 depicts an example on how this information is represented in a B-tree $\mathcal{BT}_S$[4]. Basically, a key of the B-tree is the first coordinate x of the corresponding cluster, while the value associated with the key is the pair $(color, counter)$ consisting of the color of the cluster and of the number of points which share both the same color and the same x-coordinate x (i.e., are in the represented cluster): $color = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y')$ for any $(x', y') \in \mathbb{P}$ with $x' = x$ and $counter = |\{(x', y') \mid x' = x \wedge (x', y') \in \mathbb{P}\}|$. When a tuple is inserted, we have to consider the B-tree corresponding to the compass

[3] Here we assume that the *update* of a tuple t to a tuple t' in instance r simply consists of the execution of the deletion of t in r immediately followed by insertion of t' in $r \setminus \{t\}$.

[4] In the following figures, to simplify the representation, each node (i.e., block) may contain up to 2 key-value pairs and 3 pointers to child nodes.

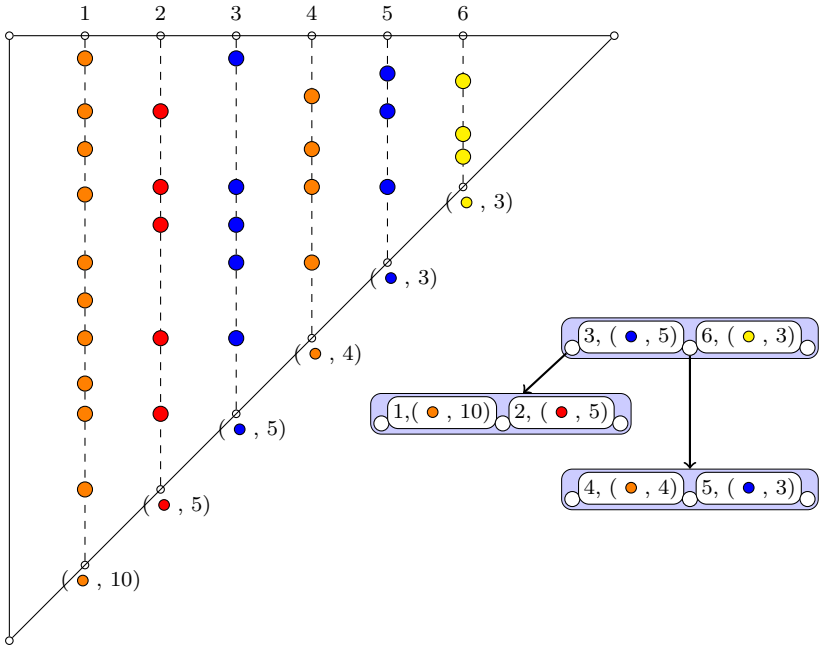

Fig. 9 A B-tree representing the needed information for keeping a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ S-consistent

structure for values of attributes X corresponding to the X values of the given tuple. On this B-tree, the coordinate x corresponding to the starting time of the tuple is checked against the keys of the B-tree. If a pair $(x, (color, counter))$ is found, we may have two different situations. If the color of the tuple (i.e., values of attributes Y) is equal to *color*, then *counter* is incremented and the tuple is allowed to be inserted into the current database instance. Otherwise, the insertion is refused as the tuple violates ITFD $X \rightarrow_S Y$. If x is not found as a key of the B-tree, then the pair $(x, (color, 1))$ is suitably inserted in the B-tree and the tuple is allowed to be inserted into the current instance. As with the tuple deletion/update, it suffices to search for the coordinate x in the right B-tree $\mathcal{BT}_S$ and consider the key-value pair $(x, (c, counter))$. If *counter* is equal to 1, then such a pair is removed from $\mathcal{BT}_S$; otherwise it is updated to $(x, (c, counter - 1))$.
The same technique may be applied to verify ITFD $X \rightarrow_F Y$. In this case, B-tree keys and values represent clusters of points on a horizontal line. That is, a key of the B-tree is the coordinate y of the corresponding cluster, while the value associated with the key is the pair $(color, counter)$ with $color = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y')$ for any $(x', y') \in \mathbb{P}$ with $y' = y$ and $counter = |\{(x', y') \mid y' = y \wedge (x', y') \in \mathbb{P}\}|$.

5.4 Verifying satisfaction of $X \rightarrow_B Y$

Checking satisfaction of $X \rightarrow_B Y$ is less easy than the previous case. First of all, we make use of four B-trees $\mathcal{BT}_S$, $\mathcal{BT}_F$, $\mathcal{BT}_{I(S)}$, $\mathcal{BT}_{I(F)}$ for each needed compass structure. For all these B-trees, the order for the whole key is lexicographic, given

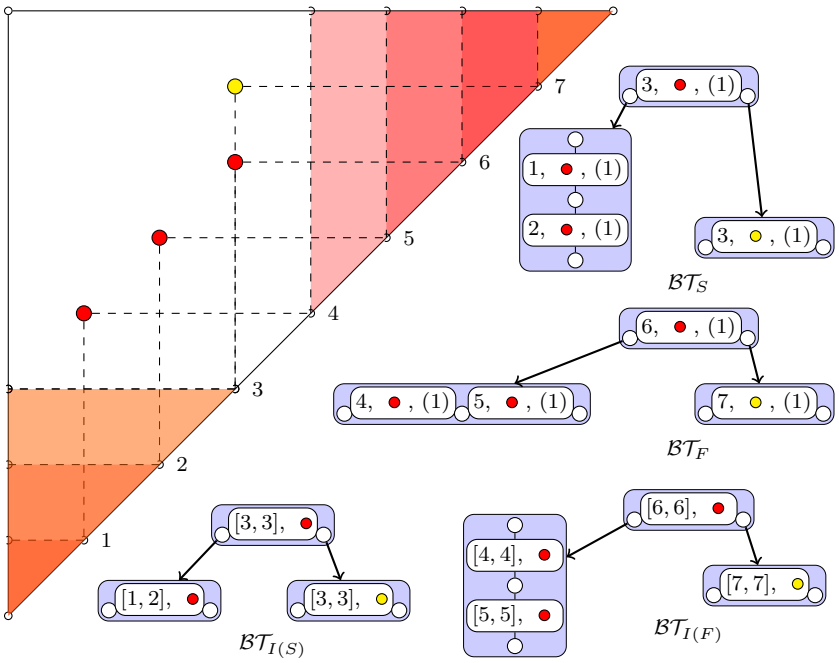

Fig. 10 An example of B-trees, needed for representing the necessary information for keeping a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ B-consistent

an arbitrary fixed-a-priori order over colors. Fig. 10 depicts an example of compass structure for 4 nodes (i.e., tuples) and the related B-trees. B-trees $\mathcal{BT}_{I(S)}$ and $\mathcal{BT}_{I(F)}$ primarily serve to keep verification time logarithmic. They store the maximal ranges for x-coordinates (i.e., starting time points) and y-coordinates (i.e., ending time points) corresponding to intervals sharing (without any "interrupting" interval) the same color, respectively. B-trees $\mathcal{BT}_S$ and $\mathcal{BT}_F$ are auxiliary structures used to efficiently update main indexes $\mathcal{BT}_{I(S)}$ and $\mathcal{BT}_{I(F)}$. B-trees $\mathcal{BT}_S$ and $\mathcal{BT}_F$ store in their key-value pairs the information about the number of points with the same color starting and finishing at a given time, respectively. The key-value pair for $\mathcal{BT}_S$ is a pair $((\bar{x}, c), \mathit{counter})$ with $\mathit{counter} = |\{(x, y) \mid x = \bar{x} \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c\}|$. In a symmetric way, the key-value pair for $\mathcal{BT}_F$ is a pair $((\bar{y}, c), \mathit{counter})$ with $\mathit{counter} = |\{(x, y) \mid y = \bar{y} \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c\}|$. The key of $\mathcal{BT}_{I(S)}$ is a range-color pair $([lb, ub], c)$ and no values need to be associated with keys for this B-tree:

- All points (x, y) with $lb < x < ub$, for every y, are colored with c;
- There exists $\overline{y}$ for which $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(lb, \overline{y}) = c$, and there exists $\overline{\overline{y}}$ for which $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(ub, \overline{\overline{y}}) = c$;
- lb is the x-coordinate for which there exists $(lb, \overline{y}) \in \mathbb{P}$ such that either lb is the minimum x-coordinate of the compass structure, or there exists x', $x' < lb$, that is the maximum point for which there exists $(x', y') \in \mathbb{P}$ for some y' with color $c' \neq c$;
- ub is the x-coordinate for which there exists $(ub, \overline{\overline{y}}) \in \mathbb{P}$ such that either ub is the maximum x-coordinate of the compass structure, or there exists x'',

$x'' > ub$, that is the minimum point for which there exists $(x'', y'') \in \mathbb{P}$ for some y'' with color $c' \neq c$.

From the definition, it turns out that ranges in the B-tree are non-intersecting (they could possibly share one endpoint). Thus, we can order them totally by looking at their start endpoints.

The definition of the key-value pair for the tree $\mathcal{BT}_{I(F)}$ is a symmetric variant of the one for $\mathcal{BT}_{I(S)}$ simply obtained using the y-coordinate. The key-value pair for $\mathcal{BT}_{I(F)}$ is a pair range-color $([lb, ub], c)$[5] (without any associated value):

- All points (x, y), with $lb < y < ub$ and for every x, are colored with c;
- There exists $\overline{x}$ for which $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(\overline{x}, lb) = c$, and there exists $\overline{\overline{x}}$ for which $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(\overline{\overline{x}}, ub) = c$;
- lb is the y-coordinate for which there exists $(\overline{x}, lb) \in \mathbb{P}$ such that either lb is the minimum y-coordinate of the compass structure, or there exists y', $y' < lb$, that is the maximum point for which there exists $(x', y') \in \mathbb{P}$ for some x' with color $c' \neq c$;
- ub is the y-coordinate for which there exists $(\overline{\overline{x}}, ub) \in \mathbb{P}$ such that either ub is the maximum y-coordinate of the compass structure, or there exists y'', $y'' > ub$, that is the minimum point for which there exists $(x'', y'') \in \mathbb{P}$ for some x'' with color $c' \neq c$.

The algorithm is quite straightforward. Suppose that we want to update the database with a tuple and that it corresponds to updating the suitable compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ with a point $p = (x, y)$ with color c:

1. We check y against $\mathcal{BT}_{I(S)}$ and we refuse the tuple if one of two cases arises: (i) there exist at least two keys $([lb', ub'], c')$ and $([lb'', ub''], c'')$ with $y < lb' \leq lb''$, so that $c' \neq c''$; (ii) there exists a key $([lb', ub'], c')$ in $\mathcal{BT}_{I(S)}$ with $y < lb'$ and $c' \neq c$;
2. We check x against $\mathcal{BT}_{I(F)}$ and we refuse the tuple if one of two cases arises: (i) there exists at least two keys $([lb', ub'], c')$ and $([lb'', y''], c'')$ with $ub'' \leq ub' < x$, so that $c' \neq c''$; (ii) there exists one key $([lb', ub'], c')$ in $\mathcal{BT}_{I(S)}$ with $ub' < x$ and $c' \neq c$;
3. If the tuple passes the two previous checks, it has to be inserted into the compass structure and the four B-trees must be updated accordingly. If there exists a node with key-value pair $((x, c), n)$ in $\mathcal{BT}_S$, then we update such a pair to $((x, c), n + 1)$; otherwise we insert $((x, c), 1)$ in $\mathcal{BT}_S$. If there exists a node with key-value pair $((y, c), n)$ in $\mathcal{BT}_F$, then we update such a pair to $((y, c), n + 1)$; otherwise we insert $((y, c), 1)$ in $\mathcal{BT}_F$. If there exists a key $([lb', ub'], c)$ in $\mathcal{BT}_{I(S)}$ with $lb' \leq x \leq ub'$, the B-tree does not need updates. If there exists a key $([lb', ub'], c')$ in $\mathcal{BT}_{I(S)}$ with $lb' \leq x \leq ub'$ and $c' \neq c$, let x_n and x_p be the coordinates in $\mathcal{BT}_S$ such that x_p is the maximum point with $lb' \leq x_p \leq x$ (i.e., no points in between x_p and x) and x_n is the minimum point with $x \leq x_n \leq ub'$ (i.e., no points in between x and x_n). We remove node $([lb', ub'], c')$ and insert nodes $([x, x], c)$, $([lb', x_p], c')$. If $lb' < ub'$, we also insert $([x_n, ub'], c')$. $\mathcal{BT}_{I(F)}$ is updated in a symmetric way using $\mathcal{BT}_F$.

[5] It is worth noting that the range $[lb, ub]$ is related in this case to the y-coordinate. Thus, $\mathcal{BT}_{I(F)}$ characterizes points (x, y) similarly to $\mathcal{BT}_{I(S)}$, but considering the y-coordinate instead of the x-one, as it is highlighted in the corresponding items.

Consider now the deletion/update operation and suppose that we want to delete a tuple corresponding to a point (x, y) with color c. First, x is checked on B-tree $\mathcal{BT}_S$ and once we have found the key-value pair $((x, c), counter)$, two cases may arise:

- If $counter > 1$, then it is updated to $((x, c), counter - 1)$;
- If $counter$ is equal to 1, then the pair is removed from $\mathcal{BT}_S$. In such a case, if a key $([x, ub'], c)$ is found in $\mathcal{BT}_{I(S)}$ and $x = ub'$, then $([x, ub'], c)$ is removed from $\mathcal{BT}_{I(S)}$. On the other hand, if $x < ub'$ for key $([x, ub'], c)$ found in $\mathcal{BT}_{I(S)}$, let $((x', c), counter')$ be the key-value pair in $\mathcal{BT}_S$ such that $x < x'$ and there is no node $((x'', c), counter'')$ with $x < x'' < x'$ (i.e., x' is the minimum x-coordinate greater than x). Key $([x, ub'], c)$ in $\mathcal{BT}_{I(S)}$ is updated to $([x', ub'], c)$. Similarly, if key $([lb', x], c)$ is found in $\mathcal{BT}_{I(S)}$ and $x = lb'$, then key $([lb', x], c)$ is removed from $\mathcal{BT}_{I(S)}$. If key $([lb', x], c)$ is found in $\mathcal{BT}_{I(S)}$ and $lb' < x$, then let $((x', c), counter')$ be the key-value pair in $\mathcal{BT}_S$ such that $x' < x$ and there is no node $((x'', c), counter'')$ with $x' < x'' < x$ (i.e., x' is the maximum x-coordinate less than x). Key $([lb', x], c)$ is updated to $([lb', x'], c)$.

The symmetric procedure is applied to $\mathcal{BT}_F$ and $\mathcal{BT}_{I(F)}$ using y.

5.5 Verifying satisfaction of $X \rightarrow_M Y$

Verifying the satisfaction of ITFD $X \rightarrow_M Y$ is similar to that of $X \rightarrow_S Y$ (, though a little bit more complex.) Given an M-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, we associate with every coordinate w having a point (w, y) or (x, w) in $\mathbb{P}$ a pair of sets $[\mathcal{C}_L^w, \mathcal{C}_R^w]$, where $\mathcal{C}_L^w = \{(c, n) \mid (x, y) \in \mathbb{P} \wedge x = w \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c \wedge n = |\{(x, y) \mid x = w \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c\}|\}$ and $\mathcal{C}_R^w = \{(c, n) \mid (x, y) \in \mathbb{P} \wedge y = w \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c \wedge n = |\{(x, y) \mid y = w \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x, y) = c\}|\}$. They keep the information for each color about the number of points with w as x-coordinate and as y-coordinate, respectively. It is easy to prove for every M-consistent compass structure that the following property holds:

Property 1 Given an M-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, for every coordinate w for which there exists a point (w, y) or a point (x, w) in $\mathbb{P}$, we have that sets $\mathcal{C}_L^w \neq \emptyset$ and $\mathcal{C}_R^w \neq \emptyset$ are both singleton sets $\mathcal{C}_R^w = \{(c, n)\}$, $\mathcal{C}_L^w = \{(c, m)\}$, and their elements share the same color.

Thus, we represent our compass structure as a B-tree $\mathcal{BT}_M$, where the key is composed by the triple $(w, direction, color)$ and the value is n, where $direction \in \{L, R\}$ and $(color, n) \in \mathcal{C}_{direction}^w$. The key values are lexicographically ordered (we assume that $L < R$ and that there is an arbitrary a-priori fixed linear order on colors). Suppose that we want to insert a point $p = (x, y)$ with color c, corresponding to a given tuple, in a M-consistent compass structure.

Then we have to look for four nodes in $\mathcal{BT}_M$ and apply the following procedure:

1. If there exist nodes $((x, R, c'), m)$ or $((y, L, c'), m)$ for some m and some $c' \neq c$ in $\mathcal{BT}_M$, then we refuse the insertion of the tuple corresponding to $p = (x, y)$ with color c;
2. If nodes considered in the previous step do not exist, then if there exists node $((x, L, c), m)$ in $\mathcal{BT}_M$, we update it to $((x, L, c), m + 1)$; otherwise we insert

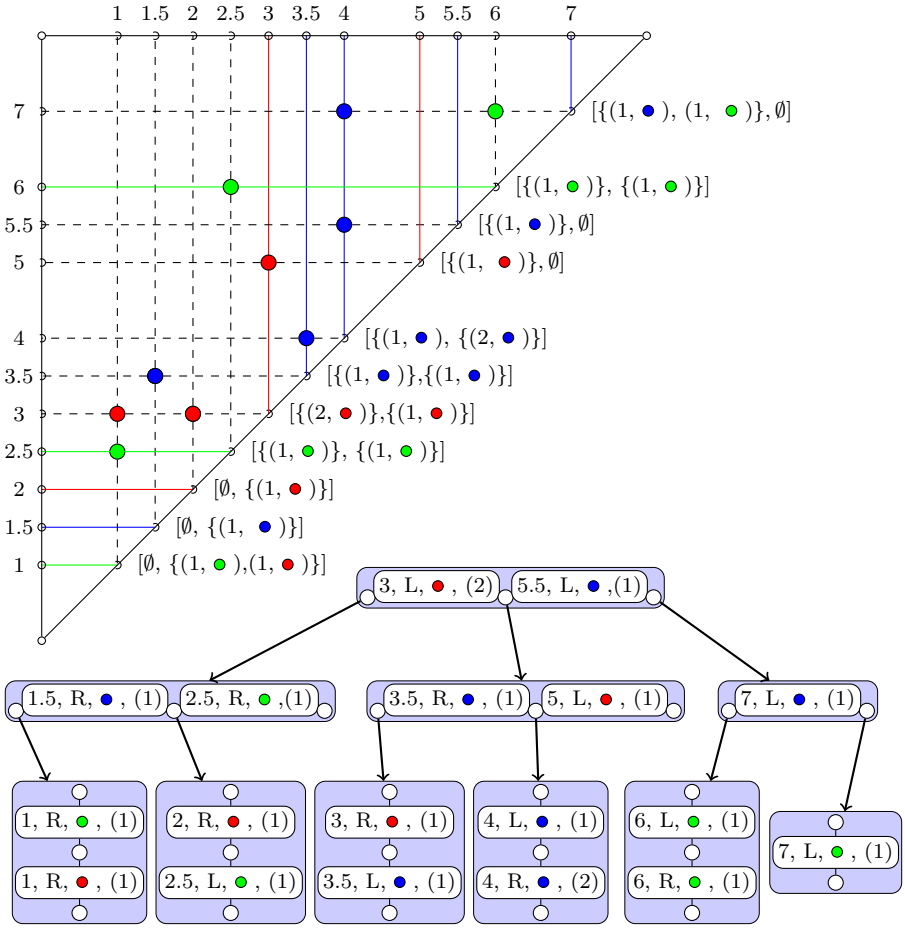

Fig. 11 A B-tree representing the needed information for keeping a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ M-consistent

node $((x, L, c), 1)$ in $\mathcal{BT}_M$. Moreover, if there exists node $((y, R, c), m)$ in $\mathcal{BT}_M$, we update it to $((y, R, c), m+1)$; otherwise we insert node $((x, R, c), 1)$ in $\mathcal{BT}_M$.

Consider now the deletion/update operation and suppose that we want to delete a tuple corresponding to a point (x, y) with color c: we have to consider nodes $((x, L, c), n)$ and $((y, R, c), m)$. If $n = 1$, we delete node $((x, L, c), n)$; otherwise we update it to $((x, L, c), n - 1)$. If $m = 1$, we delete node $((y, R, c), m)$; otherwise we update it to $((y, R, c), m - 1)$.

Fig. 11 represents a compass structure with 10 nodes, the related sets $\mathcal{C}_L^w$ and $\mathcal{C}_R^w$, and the corresponding B-tree for verifying M-consistency.

5.6 Verifying satisfaction of $X \rightarrow_D Y$

We now focus on the more complex cases of D-consistency and O-consistency and the corresponding compass structures and B-trees. For D-consistency, the shape

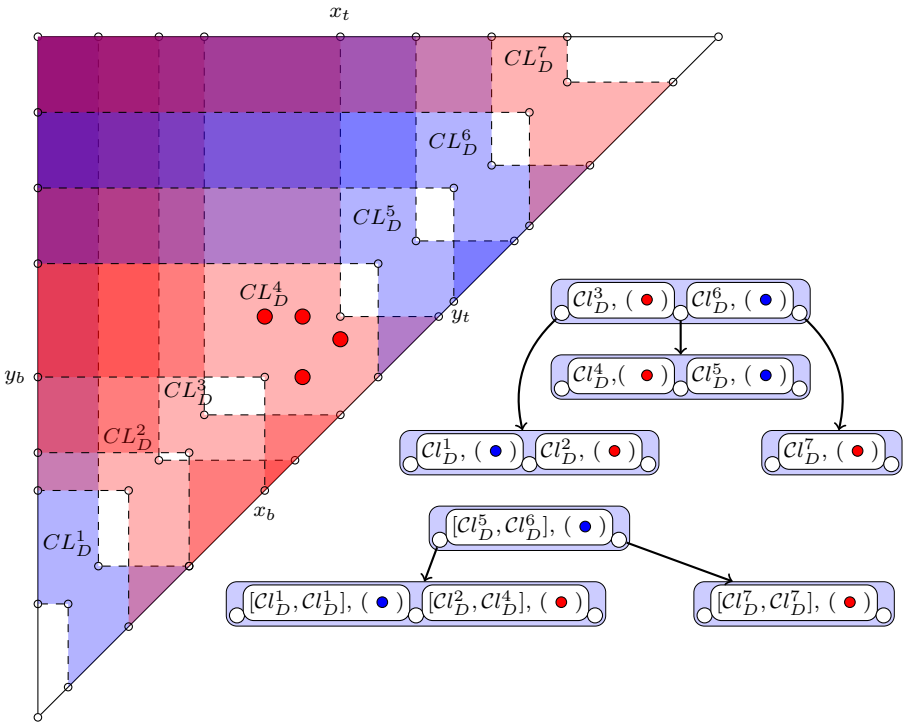

Fig. 12 B-trees representing the needed information for keeping a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ D-consistent

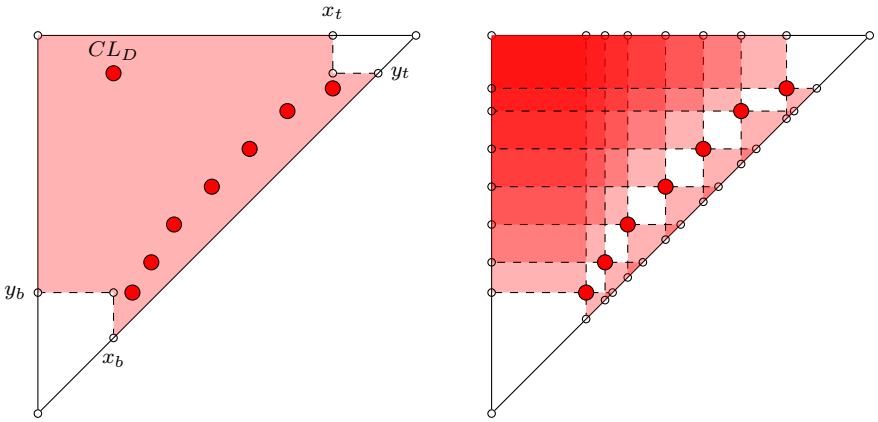

Fig. 13 An example on how delete operation may be expensive in our compass representation of clusters. If we delete the point on the top-left of the cluster we generate a number of clusters which is linear in the number of points in it

of the closure of a D-cluster $\mathcal{C}l_D$ can be expressed as

$$closure(\mathcal{C}l_D) = closure_D(\{(x_b, y_b)\}) \cup closure_D(\{(x_t, y_t)\}) \cup \\ (closure_O(\{(x_b, y_b)\}) \cap closure_O(\{(x_t, y_t)\})),$$

where $x_b = \min\{x|(x,y) \in \mathcal{C}l_D\}$, $y_b = \min\{y|(x,y) \in \mathcal{C}l_D\}$, $x_t = \max\{x|(x,y) \in \mathcal{C}l_D\}$, and $y_t = \max\{y|(x,y) \in \mathcal{C}l_D\}$.

Intuitively, to know the closure of a D-cluster, it is enough to derive the closures of two singletons, i.e., the one containing only the "synthetic" interval corresponding to (x_b, y_b), which is built using the minimum start and end of cluster intervals, and the other one corresponding to (x_t, y_t), which built using the maximum start and end of cluster intervals, respectively. We call the pair of points $gen(\mathcal{C}l_D) = ((x_b, y_b), (x_t, y_t))$ the *generation pair* for $\mathcal{C}l_D$. The shape of D-clusters is hourglass-like, as depicted in Figs. 12 and 13, where the neck corresponds to the position of the generation pair. Fig. 12 depicts the closures of 7 D-clusters. For cluster CL_D^4 the generation pair is explicitly represented together with points of the cluster.

The following property holds for generation pairs of D-clusters of a D-consistent compass structure:

Property 2 Given a D-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ and two D-clusters $\mathcal{C}l_D$, $\mathcal{C}l'_D$ of the structure with their generation pairs $gen(\mathcal{C}l_D) = ((x_b, y_b), (x_t, y_t))$ and $gen(\mathcal{C}l'_D) = ((x'_b, y'_b), (x'_t, y'_t))$, one of the following conditions holds:

- $gen(\mathcal{C}l_D) = gen(\mathcal{C}l'_D)$;
- $(x_t, y_t)B(x'_b, y'_b)$ or $(x'_t, y'_t)B(x_b, y_b)$;
- $(x_t, y_t)O(x'_b, y'_b)$ or $(x'_t, y'_t)O(x_b, y_b)$.

This property provides us with a natural total linear order $\leq_D$ between D-clusters. Formally, given a D-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ and two D-clusters $\mathcal{C}l_D$, $\mathcal{C}l'_D$ with generation pairs $gen(\mathcal{C}l_D) = ((x_b, y_b), (x_t, y_t))$ and $gen(\mathcal{C}l'_D) = ((x'_b, y'_b), (x'_t, y'_t))$, we have that $\mathcal{C}l_D <_D \mathcal{C}l'_D$ if and only if (x_t, y_t) B (x'_b, y'_b) or (x_t, y_t) O (x'_b, y'_b). Checking if a point $p = (x, y)$ belongs to $closure_D(\mathcal{C}l_D)$ of some cluster $\mathcal{C}l_D$ can be done in constant time having the generation pair $gen(\mathcal{C}l_D) = ((x_b, y_b), (x_t, y_t))$ for $\mathcal{C}l_D$. Indeed, it is enough to check whether $(x < x_t \wedge y > y_b) \vee (x > x_b \wedge y < y_t)$. Given a generation pair $gen(\mathcal{C}l_D) = ((x_b, y_b), (x_t, y_t))$ for a cluster $\mathcal{C}l_D$, we denote the point (x_b, y_b) by $\perp(\mathcal{C}l_D)$ and the point (x_t, y_t) by $\top(\mathcal{C}l_D)$.

To consistently maintain a compass structure D-consistent, we make use of B-trees $\mathcal{BT}_D$ and $\mathcal{BT}_{I(D)}$. $\mathcal{BT}_D$ is a B-tree where the keys are generation pairs representing clusters $\mathcal{C}l_D$ of $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$. Thus, they are of fixed size regardless of how many points are contained in clusters. The order between clusters is the given total order $\leq_D$. Values of both $\mathcal{BT}_D$ and $\mathcal{BT}_{I(D)}$ are the colours of clusters. $\mathcal{BT}_{I(D)}$ is a simple compact version of $\mathcal{BT}_D$. It simply stores "chunks" of consecutive clusters in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ which share the same color. A key of such a B-tree is a range $[\mathcal{C}l_D, \mathcal{C}l'_D]$ of generation pairs for two clusters with $\mathcal{C}l_D \leq_D \mathcal{C}l'_D$, while a value is a color, i.e., the one shared by all clusters between $\mathcal{C}l_D$ and $\mathcal{C}l'_D$. Since these chunks consist of consecutive clusters, the total order $\leq_D$ can be trivially adapted to them. Moreover, we want to keep these chunks maximal, that is, for every key-value pair $([\mathcal{C}l_D, \mathcal{C}l'_D], c)$ in $\mathcal{BT}_{I(D)}$, if there exists a key-value pair $([\mathcal{C}l''_D, \mathcal{C}l'''_D], c')$ where $\mathcal{C}l''_D$ is the immediate successor of $\mathcal{C}l'_D$ in $\leq_D$, then $c \neq c'$.

The main steps of the algorithm for checking whether a point $p = (x, y)$ with color c, representing the tuple to be inserted, may be inserted in a D-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, and for updating the corresponding B-trees $\mathcal{BT}_D$ and $\mathcal{BT}_{I(D)}$ when the insertion is allowed, are as follows:

1. We check the coordinates of p against clusters in $\mathcal{BT}_D$. More precisely, we do four search and obtain (if they exists) the following clusters:
   (i) $\mathcal{C}l_D^{prev}$, which is the maximum (in the order $\leq_D$) cluster such that $\top(\mathcal{C}l_D^{prev})$ B p or $\top(\mathcal{C}l_D^{prev})$ O p,
   (ii) $\mathcal{C}l_D^{next}$, which is the minimum (in the order $\leq_D$) cluster such that p B $\bot(\mathcal{C}l_D^{next})$ or p O $\bot(\mathcal{C}l_D^{next})$,
   (iii) $\mathcal{C}l_D^{min}$, which is the minimum (in the order $\leq_D$) cluster such that $p \in closure_D(\mathcal{C}l_D^{min})$,
   (iv) $\mathcal{C}l_D^{max}$, which is the maximum (in the order $\leq_D$) cluster such that $p \in closure_D(\mathcal{C}l_D^{max})$.

   The complexity of these operations corresponds to that of four simple search operations in a B-tree (i.e., logarithmic in the number of blocks).
2. If $\mathcal{C}l_D^{min}$ and $\mathcal{C}l_D^{max}$ are undefined after the previous step, then p has occurred in one of the free regions (e.g., the white rectangles in Fig. 12): thus, p can be inserted and the key-value pair $(((x,y),(x,y)),c)$ is inserted into $\mathcal{BT}_D$, representing a new cluster $\mathcal{C}l_D^p$. Now we have to consistently update $\mathcal{BT}_{I(D)}$. We check $\mathcal{C}l_D^p$ against $\mathcal{BT}_{I(D)}$ and have the following cases:
   (i) There exists two consecutive chunks $([\mathcal{C}l_D, \mathcal{C}l'_D], c')$ and $([\mathcal{C}l''_D, \mathcal{C}l'''_D], c'')$ with $\mathcal{C}l'_D <_D \mathcal{C}l_D^p <_D \mathcal{C}l''_D$. In this case, if $c' = c$, then we substitute $([\mathcal{C}l_D, \mathcal{C}l'_D], c')$ with $([\mathcal{C}l_D, \mathcal{C}l_D^p], c')$ in $\mathcal{BT}_{I(D)}$ (a symmetric operation applies if $c'' = c$), while if $c' \neq c \neq c''$, then we insert a new chunk $([\mathcal{C}l_D^p, \mathcal{C}l_D^p], c)$ in $\mathcal{BT}_{I(D)}$ between $([\mathcal{C}l_D, \mathcal{C}l'_D], c')$ and $([\mathcal{C}l''_D, \mathcal{C}l'''_D], c'')$;
   (ii) There exists $([\mathcal{C}l_D, \mathcal{C}l'_D], c')$ with $\mathcal{C}l_D <_D \mathcal{C}l_D^p <_D \mathcal{C}l'_D$. In this case, if $c' = c$, we leave $\mathcal{BT}_{I(D)}$ as it is. Otherwise, we replace $([\mathcal{C}l_D, \mathcal{C}l'_D], c')$ in $\mathcal{BT}_{I(D)}$ with three consecutive key-value pairs $([\mathcal{C}l_D, \mathcal{C}l_D^{prev}], c')$, $([\mathcal{C}l_D^p, \mathcal{C}l_D^p], c)$, and $([\mathcal{C}l_D^{next}, \mathcal{C}l'_D], c')$.
3. If $\mathcal{C}l_D^{min}$ and $\mathcal{C}l_D^{max}$ are defined, then p occurs almost in a cluster. If the color of $\mathcal{C}l_D^{min}$ and the color of $\mathcal{C}l_D^{max}$ are both equal to c, we proceed; otherwise, we refuse the insertion of the corresponding tuple. We then check $[\mathcal{C}l_D^{min}, \mathcal{C}l_D^{max}]$ against $\mathcal{BT}_{I(D)}$: if there exists a chunk $([\mathcal{C}l_D, \mathcal{C}l'_D], c)$ with $\mathcal{C}l_D \leq_D \mathcal{C}l_D^{min} \leq_D \mathcal{C}l_D^{max} \leq_D \mathcal{C}l'_D$, then we proceed; otherwise, we refuse the insertion of the corresponding tuple. Suppose that such a chunk does exist and let $([\overline{\mathcal{C}l}_D, \overline{\mathcal{C}l}'_D], c')$ and $([\overline{\mathcal{C}l}''_D, \overline{\mathcal{C}l}'''_D], c'')$ be its immediate predecessor and successor, respectively: if $(\top(\overline{\mathcal{C}l}'_D)\ B\ p \vee \top(\overline{\mathcal{C}l}'_D)\ O\ p) \wedge (p\ B\ \bot(\overline{\mathcal{C}l}''_D) \vee p\ O\ \bot(\overline{\mathcal{C}l}''_D))$, then the tuple can be inserted; otherwise, it is refused. Moreover, let $\bot(\mathcal{C}l_D^{min})$ be the point (x_{min}, y_{min}) and $\top(\mathcal{C}l_D^{max})$ be the point (x_{max}, y_{max}): if p is inserted, we have to merge all clusters from $\mathcal{C}l_D^{min}$ to $\mathcal{C}l_D^{max}$ into a single node we name $\mathcal{C}l_D^{new}$. It causes the merging of consecutive key-value pairs in $\mathcal{BT}_D$. Indeed, all the key-value pairs between $\mathcal{C}l_D^{min}$ and $\mathcal{C}l_D^{max}$ are deleted and the new key-value pair $(((min(x, x_{min}), min(y, y_{min})), (max(x, x_{max}),\ max(y, y_{max}))), c)$ is inserted in their place[6]. Finally, we update the chunk $([\mathcal{C}l_D, \mathcal{C}l'_D], c)$ in $\mathcal{BT}_{I(D)}$ by replacing $\mathcal{C}l_D$ with $\mathcal{C}l_D^{new}$, if and only if $\mathcal{C}l_D = \mathcal{C}l_D^{min}$, and by replacing $\mathcal{C}l'_D$ with $\mathcal{C}l_D^{new}$, if and only if $\mathcal{C}l'_D = \mathcal{C}l_D^{max}$.

For the deletion of a point (corresponding to a tuple), it is worth noting that in the worst case this operation could lead to the creation of a linear number of

[6] The operation of merging D-clusters implies the classic range deletion in a B-tree which can be done in an efficient (logarithmic) way [14, 15].

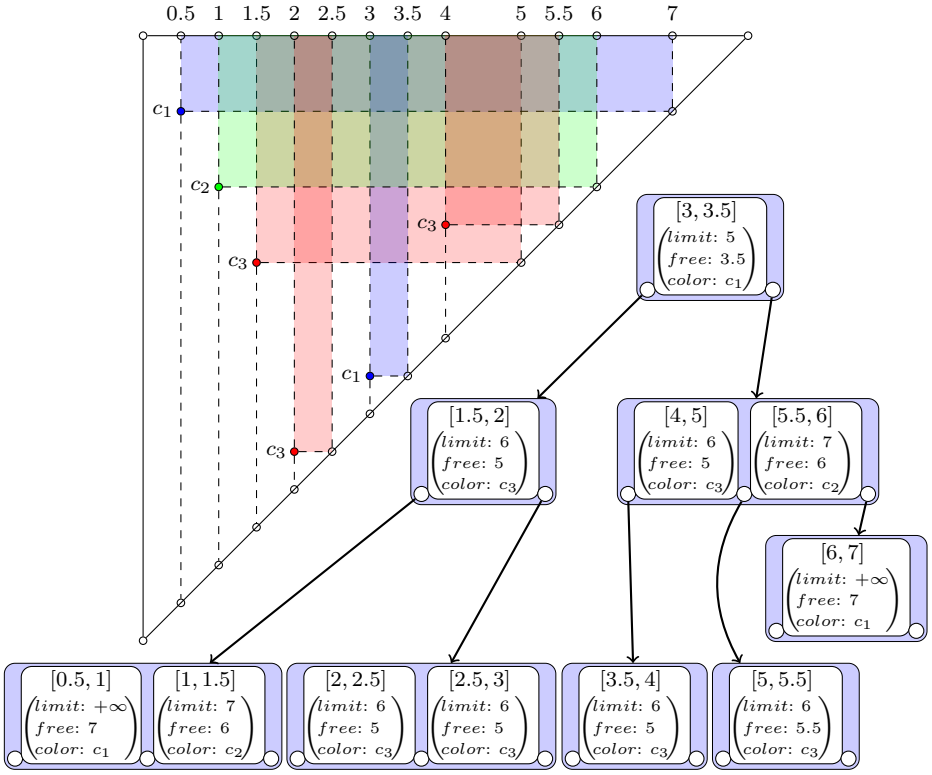

Fig. 14 A B-tree representing the needed information for (partially) keeping a compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ O-consistent: only $\mathcal{BT}_V$ is depicted and the corresponding slices in the corresponding compass structure are highlighted

clusters. As an example, Fig. 13 depicts a D-consistent compass structure and the induced D-cluster. If the tuple corresponding to the top-left point needs to be deleted, we have to represent in the suitable B-trees a cluster for each of the remaining points as highlighted in the figure.

5.7 Verifying satisfaction of $X \rightarrow_O Y$

To check whether an O-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$ may be updated with a point $p = (x, y)$ having color c and representing a given tuple, we make use of two B-trees $\mathcal{BT}_V$ and $\mathcal{BT}_H$. Informally, when a new tuple $p = (x, y)$ with color c is inserted, $\mathcal{BT}_V$ contains the needed data to check whether the tuple is consistent with the tuples that overlap it. More precisely, we check that, for every $(x', y') \in \mathbb{P}$ with $x' < x < y' < y$, condition $\mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y') = c$ holds. An example of $\mathcal{BT}_V$ is depicted in Fig. 14. There we have to check the rectangles that contain the new point and correspond to intervals overlapping the interval represented by this point.

Given an O-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, we define a (vertical) *compass-slice* to be a pair of points $[lb, ub]$ with $lb < ub$, such that there exist two points

(x', y'), $(x'', y'') \in \mathbb{P}$ where $(x' = lb \vee y' = lb) \wedge (x'' = ub \vee y'' = ub)$ and, for each point $(\overline{x}, \overline{y}) \in \mathbb{P}$, it holds $\overline{x} \leq lb < ub \leq \overline{y}$. Basically, a compass slice is a pair of consecutive coordinates (i.e., no other point coordinates in between) in the set collecting both the first and the second coordinates for all points in $\mathbb{P}$. Each vertical compass-slice may be characterized by three parts:

- the first one where a point can be added, no matter what its color is,
- the second one where a point can be added only if it shares the same color of already existing points,
- and the third one where no points can be added, as they would be in an area where points with different colours have to be considered.

These three parts correspond to three different situations:

(i) The tuple corresponding to the new point is not overlapped by any other tuple, i.e., point of the structure.
(ii) The tuple corresponding to the new point is overlapped by other tuples corresponding to points with same color (i.e., the same values for attributes Y).
(iii) The tuple corresponding to the new point is overlapped by other tuples corresponding to points with different colours (i.e., different values for attributes Y), respectively.

A similar approach has to be taken for tuples that the new tuple overlaps. This is done by using a the B-tree $\mathcal{BT}_H$ which concerns horizontal compass-slices. In the following, we will focus only on how to efficiently check consistency with respect to $\mathcal{BT}_V$ for points overlapped by the new point. The same operations can be translated in a symmetric way to deal with $\mathcal{BT}_H$.

Given a compass slice $[lb, ub]$, we define its *color* c, its *free* bound f and its *limit* l as follows:

- $f = \min\{y' | (x', y') \in \mathbb{P} \wedge x' \leq lb \wedge y' \geq ub\}$;
- $l = \min\{y' | (x', y') \in \mathbb{P} \wedge x' \leq lb \wedge y' > ub \wedge \exists (x'', y'') \in \mathbb{P}(x'' \leq lb \wedge ub \leq y'' < y' \wedge \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x'', y'') \neq \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y'))\}$
- $c = \mathcal{G}_{(\mathbb{P},\mathcal{C})}(x', y')$ where $(x', y') \in \mathbb{P} \wedge x' \leq lb \wedge y' \geq ub \wedge y' = \min\{y'' | (x'', y'') \in \mathbb{P} \wedge x'' \leq lb \wedge y'' \geq ub\}$.

When a point (x, y) is inserted, we look for the (unique) vertical slice $[lb', ub']$, which contains it ($lb' \leq x \leq ub'$): the value of f for such slice determines the maximum value that y may have in order to be consistent even if its color c is different from the color c' of the slice. If $y > f$, it must be $c = c'$. In any case, $y \leq l$ because above l every insertion violates the consistency with respect to the given ITFD since the point occurs in the intersection of two rectangles with different colours. It would mean that there were two tuples of different colours both overlapping the new tuple. We note that c and f always exist for every compass slice $[lb, ub]$, but l might not be defined (as in the case of slices $[0.5, 1]$ and $[6, 7]$ in Fig. 14). In such cases we put $l = +\infty$. We store the compass slices in $\mathcal{BT}_V$. Each key is the corresponding slice itself $[lb, ub]$ (the order is the underlying linear order, since compass slices by definition do not intersect), and the value is the triple (l, f, c).

Now consider the following property:

Property 3 Given an O-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, if a tuple corresponding to point (x, y) with color c is allowed to be inserted in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, then (i) either there is no vertical compass slice $[lb', ub']$ with $lb' < x < ub'$, (ii) or there exists a vertical compass slice $[lb', ub']$ with color c', free bound f and limit l, where $lb' < x < ub'$ and either $y < f$ or $c' = c \wedge y < l$.

This property gives the necessary but not sufficient condition to allow a tuple to be inserted in an O-consistent compass structure. Indeed, we can define the symmetric condition for horizontal slices (stored in $\mathcal{BT}_H$):

Property 4 Given an O-consistent compass structure $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, if a tuple corresponding to point (x, y) with color c is allowed to be inserted in $\mathcal{G}_{(\mathbb{P},\mathcal{C})}$, then (i) either there is no horizontal compass slice $[lb', ub']$ with $lb' < y < ub'$, (ii) or there exists a horizontal compass slice $[lb', ub']$ with color c', free bound f and limit l, where $lb' < y < ub'$ and either $x > f$ or $c' = c \wedge x > l$.

Only with both these properties we do have a necessary and sufficient condition to establish whether a tuple may be inserted in an O-consistent compass structure.

The insertion of a new tuple corresponding to point (x, y) with color c may affect more than just the slice (if any) $[lb', ub']$ with $lb' < x < ub'$. Consider, for example, the compass structure depicted in Fig. 14 and suppose we have all points in $\mathcal{BT}_V$ but the c_1-coloured point $(0.5, 7)$. What happens if we want to insert the tuple corresponding to such point? Since it does not violate any constraint, we can insert it. Such an insertion is critical from the point of view of the complexity. Indeed, it may modify the limit l and the free bound f of all nodes in the tree. In a naive way, one may be tempted to update the whole tree. However, it turns out that slices that may be affected by the update are all contiguous, i.e., without "holes" between them. In order to deal efficiently with such cases, we simply allow pointers to have an (eventually empty) label consisting of a color c, a limit l, and a free bound f, exactly like nodes in B-Tree $\mathcal{BT}_V$[7]. If we have a pointer to a subtree of $\mathcal{BT}_V$ whose nodes are all affected by the new insertion, it suffices to label this pointer with the triple relative to the newly inserted tuple and to take it into account when verifying subsequent insertions (we shortly show how this is done). In the following we will see how this modified B-tree works.

5.7.1 Searching through a $\mathcal{BT}_V$ and updating it

Since the standard machinery of B-trees has been slightly modified, we show first how to search through a $\mathcal{BT}_V$, taking into account labels on edges. We then explain how to consistently update labels for edges involved in the standard splitting operation of a B-tree. Search through $\mathcal{BT}_V$ for a suitable tuple insertion is straightforward and the modified algorithm is shown in Fig. 15. It is the standard B-tree search enhanced with an additional check on edge labels. If the tuple satisfies all conditions on key-value pairs and edge labels, the slice (if any) is returned and the tuple is accepted for insertion; otherwise, the tuple is refused.

When a node must be split due to the excessive number of its key-value pairs, then its median key-value pair is lifted to the parent node and the node is split. Basically, the two pointers that surrounded the median node before the splitting

[7] For sake of simplicity in Fig. 14 we represent a $\mathcal{BT}_V$ without specifying labels for pointers.

remain unchanged in their labels and become the two extremes (one the smallest and one the biggest) of the new nodes. The pointer to the old node is duplicated and the two copies are the surrounding pointers for the lifted key-value pair in the father node, as depicted in Fig. 16. Suppose that we have checked a tuple, corresponding to a point (x, y) with color c, against both B-trees $\mathcal{BT}_V$ and $\mathcal{BT}_H$ and no constraints are violated. We describe how to insert it in $\mathcal{BT}_V$ (the insertion in $\mathcal{BT}_H$ is symmetric). The procedure is done in two phases.

- In the first one, we insert the new slices and modify slices $[lb', ub']$ and $[lb'', ub'']$, if any, for which $lb' < x < ub'$ and $lb'' < y < ub''$.
- In the second phase, after these insertions, we update the constraints on edges and slices affected by the presence of the new point.

For the first phase we consider only the case in which both $[lb', ub']$ and $[lb'', ub'']$ exist with values (l', f', c') and (l'', f'', c''), respectively (the remaining cases are simpler and straightforward). Two cases may arise:

- $[lb', ub'] = [lb'', ub'']$: in this case, we have that only one slice is "broken". We update $[lb', ub']$ to $[x, y]$, c' to c, l' to f' if $c \neq c'$ (otherwise unchanged), and f' to y. Then we insert two pairs with keys $[lb', x]$ and $[y, ub']$, respectively, with the same value (l', f', c');
- $[lb', ub'] \neq [lb'', ub'']$: in this case, we have $ub' \leq lb''$. We update $[lb', ub']$ to $[x, ub']$, c' to c, and l' to f' if $c \neq c'$ (otherwise unchanged). Moreover, we update $[lb'', ub'']$ to $[lb'', y]$, c'' to c, and l'' to f'' if $c \neq c''$ (unchanged otherwise). Finally, we insert two new key-value pairs $[lb', x]$, with value (l', f', c'), and $[y, ub'']$, with value (l'', f'', c'').

Now the slices are updated and we then update the constraints on edges by executing the algorithm shown in Fig. 17. It exploits labeled pointers to avoid the exploration of the whole tree, as discussed for the example of Fig. 14. Note that in the worst case there will be at most two descents in the B-tree affected by constraint updates. Thus the cost remains logarithmic in the number of disk-access operations. Deletion presents the same (worst case) difficulties we discussed for the ITFD with relation D.

The goal of this algorithm is to update the key-value pairs of both nodes and edge labels of B-tree $\mathcal{BT}_V$ to guarantee that the next tuple insertions can consider, if needed, the further constraint required by the tuple just inserted, which corresponds to point (x, y) with color c. Let n be the current node. The algorithm starts by exploring the tree from the root r ($n = r$). Let $[lb_1, ub_1], \ldots, [lb_{size(n)}, ub_{size(n)}]$ be the sequence of keys of n according to the order used by $\mathcal{BT}_V$ ($ub_i \leq lb_{i+1}$ for each $i = 1, \ldots, size(n) - 1$). Let $P_1, \ldots, P_{size(n)+1}$ be the sequence of the corresponding pointers to subtrees (P_i refers to a subtree with keys $[lb_j, ub_j]$, where $ub_{i-1} \leq lb_j \leq ub_j \leq lb_i$). If $y \leq lb_1$, the algorithm considers the subtree with slices where both bounds are less than lb_1 (first **if** instruction) using the corresponding pointer P_1 (pointing to the root of the considered subtree). By symmetry, if $ub_{size(n)} \leq x$, the algorithm considers the subtree with slices where both bounds are greater than $ub_{size(n)}$ (last **if** instruction), using the corresponding pointer $P_{size(n)+1}$. If the two preceding conditions are false, pointers and key-value pairs are considered according to the given order.

When the algorithm is running, slices have been already updated with values x and y and so there does not exist an index i, $i = 1, \ldots, size(n) - 1$, for which

Algorithm 5.1: OB-TREE-SEARCH$(n, [x, y], c)$

$i \leftarrow 1$
while $i \leq size(n) \wedge ub_i < x$
  do $i \leftarrow i + 1$
if $i \leq size(n) \wedge lb_i < x < ub_i$
  then { **if** $(y < f_i \vee (c = c_i \wedge y < l_i))$
    then return $([lb_i, ub_i], (l_i, f_i, c_i))$
    else return (**false**)
if $leaf(n)$
  then return $(null)$
  else if $y < free(P_i) \vee (c = color(P_i) \wedge y < limit(P_i))$
  then
  do { $n \leftarrow DISK - READ(P_i)$
    return $(OB\text{-}TREE\text{-}SEARCH(n, [x, y], c))$

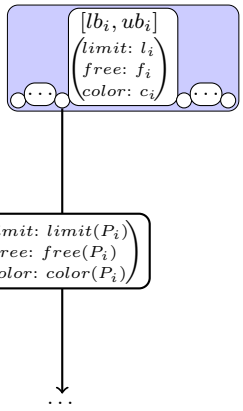

Fig. 15 The procedure for searching through a $\mathcal{BT}_V$

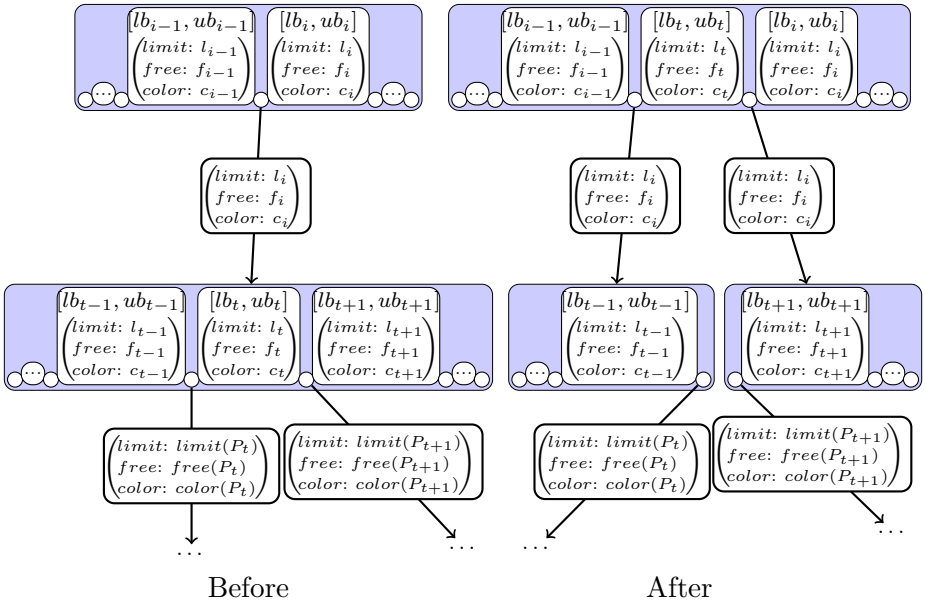

Fig. 16 An example of splitting a $\mathcal{BT}_V$

$lb_i < x < ub_i$ or $lb_i < y < ub_i$. This reduces the possible cases to consider. Turning our focus on the **for**-cycle of the algorithm, if lb_i is greater than or equal to x and ub_i is less than or equal to y, then slice $[lb_i, ub_i]$ may be modified by the constraint introduced by the last inserted tuple. In this case, we have to locally verify it as in the following:

– If $y < l_i$ and $c \neq c_i$, the new limit l_i for the current slice becomes the greatest value between y and f_i.

Algorithm 5.2: OB-TREE-CONSTRAIN$(n, [x, y], c)$

if $y \leq lb_1 \wedge \neg leaf(n)$
  then $\begin{cases} n' \leftarrow DISK\text{-}READ(P_1) \\ OB\text{-}TREE\text{-}CONSTRAIN(n', [x, y], c) \end{cases}$
for $i \leftarrow 1$ **to** $size(n)$
  do $\begin{cases} \textbf{if } x \leq lb_i \wedge y \geq ub_i \\ \quad \textbf{then} \begin{cases} \textbf{if } y < l_i \wedge c \neq c_i \\ \quad \textbf{then } l_i = max(y, f_i) \\ \textbf{if } (y < f_i) \\ \quad \textbf{then} \begin{cases} c_i = c \\ f_i = y \end{cases} \end{cases} \\ \textbf{if } x \leq ub_i \wedge y \geq lb_{i+1} \\ \quad \textbf{then} \begin{cases} \textbf{if } y < limit(P_{i+1}) \wedge c \neq color(P_{i+1}) \\ \quad \textbf{then } limit(P_{i+1}) = \max(y, free(P_{i+1})) \\ \textbf{if } (y < free(P_{i+1})) \\ \quad \textbf{then} \begin{cases} color(P_{i+1}) = c \\ free(P_{i+1}) = y \end{cases} \end{cases} \\ \textbf{if } (ub_i < y < lb_{i+1} \vee ub_i < x < lb_{i+1}) \\ \quad \textbf{then} \begin{cases} n' \leftarrow DISK\text{-}READ(P_{i+1}) \\ OB\text{-}TREE\text{-}CONSTRAIN(n', [x, y], c) \end{cases} \end{cases}$
if $ub_{size(n)} < x \wedge \neg leaf(n)$
  then $\begin{cases} n' \leftarrow DISK\text{-}READ(P_{size(n)+1}) \\ OB\text{-}TREE\text{-}CONSTRAIN(n', [x, y], c) \end{cases}$

Fig. 17 The procedure for updating a $\mathcal{BT}_V$

– If y is less than f_i, slice color c_i is updated to the color associated with (x, y) and f_i takes value y.

If $x \leq ub_i$ and $y \geq lb_{i+1}$, then the subtree referenced by P_{i+1} contains slices with keys $[lb', ub']$ for which $x \leq lb' \leq ub' \leq y$. Therefore, we should consider all nodes of the subtree, as we did for the slices of the current node. Instead of this expensive (in the number disk accesses) operation, we modify the label of pointer P_{i+1} as we did with the suitable key-value pairs of the current node. Modifying the pointer label does not require any further disk access and is enough to guarantee that the new constraint will be considered in the next tuple insertions, until a new stronger constraint ultimately supersedes it.

The last **if** nested in the **for**-cycle represents the case where x and/or y are between slices $[lb_i, ub_i]$ and $[lb_{i+1}, ub_{i+1}]$. It happens only if $ub_i < lb_{i+1}$, so we are in an internal node since slices in any leaf node are contiguous. In this case, we recursively call the algorithm on the node pointed by P_{i+1}. To evaluate the computational complexity of the algorithm, we have to take into account that the three recursive calls are mutually exclusive. That is, for a given node n, the algorithm is either called in the first **if** or in the last one or (possibly several times) in the **for**-cycle. The recursive call in the **for**-cycle may occur at most two times for the same node: a first time for x satisfying the condition and with a corresponding pointer $P_{\overline{x}}$, and a second time for y satisfying the condition and with pointer $P_{\overline{y}}$,

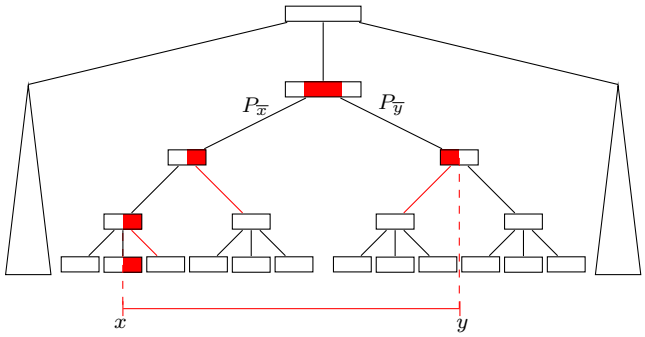

Fig. 18 Nodes and pointers (highlighted in red) whose value may be affected by the propagation of the constraint introduced by the insertion of a new point (x, y) with color c

and $\overline{x} < \overline{y}$. In this case, two different paths in the B-tree would be considered by the algorithm. That may happen only for one (internal) node (the upper partially filled node in the tree depicted in Fig. 18). In the nodes considered in Fig. 18, y will be always greater that any bound of a slice in the subtree pointed by $P_{\overline{x}}$, while x will be always less than any bound of any slice in the subtree pointed by $P_{\overline{y}}$. Thus, in the worst case the number of accessed nodes will be twice as much as the height of the B-tree (which is logarithmic with respect to the number of nodes).

5.7.2 Dealing with the motivating example from clinical medicine

Let us now consider how to deal with insertion of tuples in the instance of *Path-Therapies* depicted in Fig. 1. We will consider ITFD $PatId \rightarrow_O Phys$, we already introduced and motivated in Sect. 3 and Sect. 4. Figs. 19, 20, 21, and 22 depict B-trees corresponding to the instance in Fig. 1, assuming that tuples are inserted according to the order induced by column #. For the patient with $PatId = 1$, Figs. 19 and 20 depict B-tree $\mathcal{BT}_V$ before and after the insertion of tuple #11, where $PatId = 1$, $Phys = Dorian$, $B = 1$ and $E = 18$, respectively.

For the compass structure associated with the patient with $PatId = 1$ represented through B-trees $\mathcal{BT}_H$ and $\mathcal{BT}_V$, when we insert tuple #11 we need to verify whether it is possible to insert point $(1, 18)$ with color *Dorian* and still maintain an O-consistent compass structure (i.e., the corresponding instance satisfies ITFD $PatId \rightarrow_O Phys$). The verification is performed by looking for the key $[lb_i, ub_i]$ in $\mathcal{BT}_V$ with $lb_i \leq 1 \leq ub_i$ and comparing the corresponding values f_i, l_i with 18 and c_i with *Dorian*. In our case, the considered *slice* has key $[1, 2]$ and values $l = +\infty$, $f = 16$, and $c = Dorian$. Thus, the insertion is fine with $\mathcal{BT}_V$. A symmetric approach must be followed by looking for key $[lb_j, ub_j]$ in $\mathcal{BT}_H$ with $lb_j \leq 18 \leq ub_j$ and comparing the corresponding values f_j, l_j with 1 and c_j with *Dorian*. Even in this case, the insertion is fine with $\mathcal{BT}_H$. Thus, the tuple can be inserted as it does not violate ITFD $PatId \rightarrow_O Phys$. The updated $\mathcal{BT}_V$ and $\mathcal{BT}_H$ are depicted in Fig. 20 and Fig. 21, respectively. It is interesting to observe that the x-coordinate 1 of the new tuple belongs to the leftmost slice $[1, 2]$ in $\mathcal{BT}_V$, while the y-coordinate 18 of the new tuple belongs to slice $[18, 19]$ in the rightmost node of $\mathcal{BT}_V$. Thus, in a naive approach we should update each node of the B-tree

with the new induced constraint. Instead, we only modify nodes in the path from the root to the node containing the slice related to the x-coordinate and nodes in the path from the root to the node containing the slice related to the y-coordinate. Indeed, as depicted in Fig. 20, nodes related to slices $[8, 9]$, $[10, 11]$, and $[13, 15]$ are not modified. Only labels of edges pointing to these nodes are modified as this is does not require any further disk access since these labels are stored in the father node.

We finish our discussion of O-consistent compass structures and related B-trees $\mathcal{BT}_H$ and $\mathcal{BT}_V$ by showing for the instance of *PathTherapies* why both B-trees $\mathcal{BT}_H$ and $\mathcal{BT}_V$ are needed to verify the satisfaction of the corresponding O-based ITFD. Let us assume that point $(17, 19)$ is coloured by $c \neq Dorian$. In other words, suppose that tuple #6 in Fig. 1 has a value different from *Dorian* for attribute *Phys*. It is easy to observe in Fig. 21 that if we changed the color of point $(17, 19)$, then the compass structure after the insertion of tuple #11 would not be O-consistent. From the standpoint of the checking algorithm, the verification on B-tree $\mathcal{BT}_V$ of Fig. 20 is still successful. In contrast, checking B-tree $\mathcal{BT}_H$ would highlight a constraint violation. Indeed, the compass structure $\mathcal{BT}_H$ would contain a slice $[17, 19]$ with $l = 0$, $f = 16$, and $c \neq Dorian$, and so we would have $17 < 18 < 19$, $1 < f$, and $c = Dorian$, inducing a constraint violation on the $\mathcal{BT}_H$ (cf. Property 4). Intuitively, $\mathcal{BT}_V$ is used to verify the consistency of all tuples holding on intervals that overlap with the interval of the inserted tuple. In our case, there are no such tuples overlapping with the inserted one and thus there is no constraint violation for $\mathcal{BT}_V$. Similarly, $\mathcal{BT}_H$ is used to verify the consistency of all tuples holding on intervals the interval of the inserted tuple overlaps with. In our case, tuples corresponding to points $(1, 18)$ and $(17, 19)$ overlap and have different colors and thus there is a constraint violation for $\mathcal{BT}_H$. Let us now consider tuples related to patient with $PatId = 2$, depicted in Fig. 1. Now $PatId$ is in the antecedent of ITFD $PatId \rightarrow_O Phys$, so we have a different compass structure for verifying O-consistency with respect to the given ITFD. The compass structure considered and the corresponding B-tree $\mathcal{BT}_V$ are shown in Fig. 22, before the insertion of tuple #10, with $PatId = 2$, $Phys = Reid$, $B = 8$ and $E = 14$. Such an insertion would produce an instance violating ITFD $PatId \rightarrow_O Phys$. Indeed, slice $[7, 9]$ has the same color $c = Reid$, but would not produce a consistent $\mathcal{BT}_V$ because E, which corresponds to the y-coordinate of the point we are considering, is greater than the *limit* of slice $[7, 9]$, being $E = 14$ and $limit = 10$. Indeed, the inserted tuple corresponds to an interval overlapped by the interval corresponding to a tuple with attribute $Phys = Kelso$, thus violating ITFD $PatId \rightarrow_O Phys$.

6 Discussion and conclusions

In this paper we have proposed a set of interval-based temporal functional dependencies to express interval-based integrity constraints on temporal relational data. We have discussed its expressiveness by means of some examples taken from the clinical domain. Moreover, for each interval relation $\sim$ we have proposed suitable data structures, based on the widely known B-trees and an original spatial representation of tuples through compass structures, with associated algorithms to efficiently verify both in a static and in an incremental way whether a temporal database instance r satisfies an ITFD based on $\sim$. We have shown that

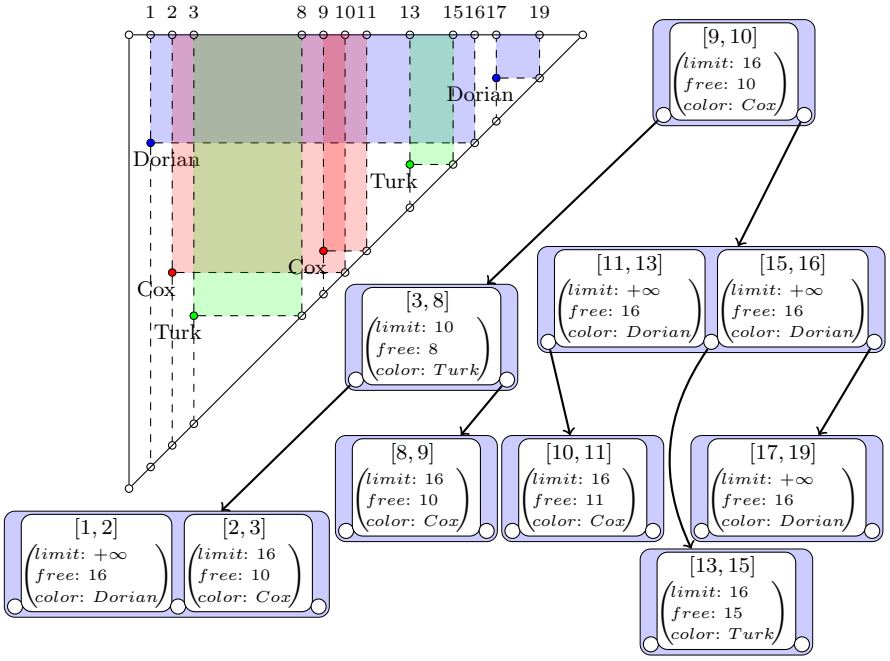

Fig. 19 The compass structure and the corresponding $\mathcal{BT}_V$ for managing ITFD $PatId \rightarrow_O Phys$ with $PatId = 1$ for the relation in Fig. 1 and before the insertion of tuple #11

the proposed ITFDs can express temporal FDs not expressible by point-based TFDs [2,9,32,36]. Even though some normalizations have been proposed based on (point-based) TFDs, TFDs have been mainly used as a way of expressing temporal constraints on temporal relational databases [36]. Therefore, we have focused here on the specification of (new) interval-based temporal constraints and on new algorithms for the efficient verification of ITFD satisfaction. Thus, both the derivation of all dependencies from a given initial set of TFDs, and the use of such TFDs for normalization of temporal database schemata need further research.

A further planned step towards interval-based temporal functional dependencies will be devoted to extending the proposed ITFDs and algorithms to deal with multiple temporal granularities and with interval-based tuple evolutions, similar to the Vianu's (point-based) approach. For example, for multiple granularities as defined by Bettini et al. [2], we have to consider that intervals come into play even to represent granules. By interpreting granules as intervals, we can express granularity-based ITFDs in a coherent way with the proposed temporal data model. That is, a database instance r of $\mathcal{R}$ satisfies an ITFD $X \rightarrow^{\sim^1}_{G,\sim^2} Y$ if for each pair of tuples t_1 and t_2 such that $[t_1[B], t_1[E]] \sim^1 [t_2[B], t_2[E]]$, there exist a granule g of granularity G where $[t_1[B], t_1[E]] \sim^2 g \wedge [t_2[B], t_2[E]] \sim^2 g$ holds, and $t_1[X] = t_2[X]$, then it is also true that $t_1[Y] = t_2[Y]$. For example, the constraint "*overlapping drug therapies within the same month for a given patient must have the same physician*" on our schema *PatTherapies* could be expressed by ITFD $PatId \rightarrow^{O}_{Month,D} Phys$. Using the approach sketched here, we plan to

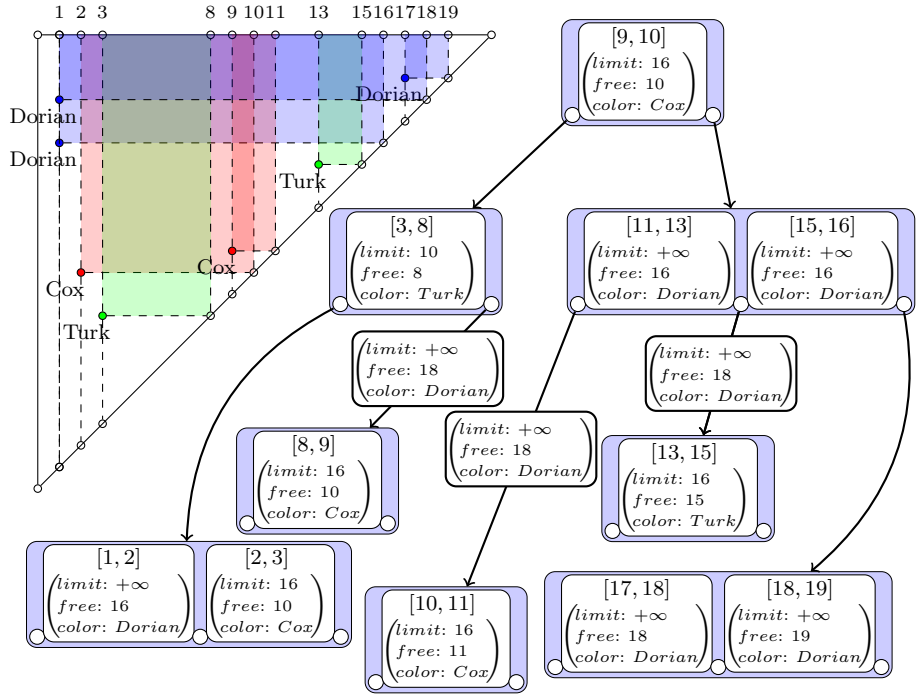

Fig. 20 The compass structure and the corresponding $\mathcal{BT}_V$ for managing ITFD $PatId \rightarrow_O Phys$ with $PatId = 1$ for the relation in Fig. 1 after the insertion of tuple #11 with $PatId = 1, Phys = Dorian$ and interval $[1, 18]$

extend the proposed ITFDs to even more general granularities. Algorithms need to be suitably extended to support this.

For the clinical domain, we plan to implement the proposed algorithms and evaluate them in different real-world clinical scenarios. Moreover, we plan to adapt and extend the proposed techniques to data mining. Indeed, in the clinical domain there is the need of mining temporal association rules among temporal data, which is often characterized by intervals of validity. An ITFD may be regarded as a kind of interval-based temporal association rule.

Acknowledgments

We would like to thank the anonymous reviewers who helped us with their constructive comments to reach this final version of our paper. We would also thank guest editor Ben Moszkowski for his great care following the review phase by offering many important and appreciated suggestions.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)

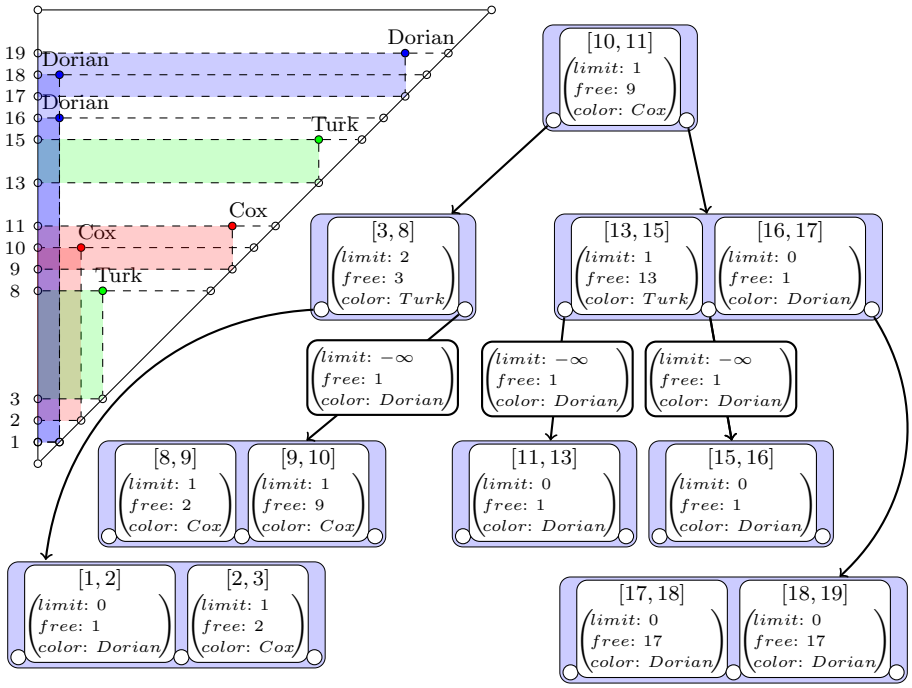

Fig. 21 The compass structure and the corresponding $\mathcal{BT}_H$ for managing ITFD $PatId \rightarrow_O Phys$ with $PatId = 1$ for the relation in Fig. 1 after the insertion of tuple #11 with $PatId = 1, Phys = Dorian$ and interval $[1, 18]$

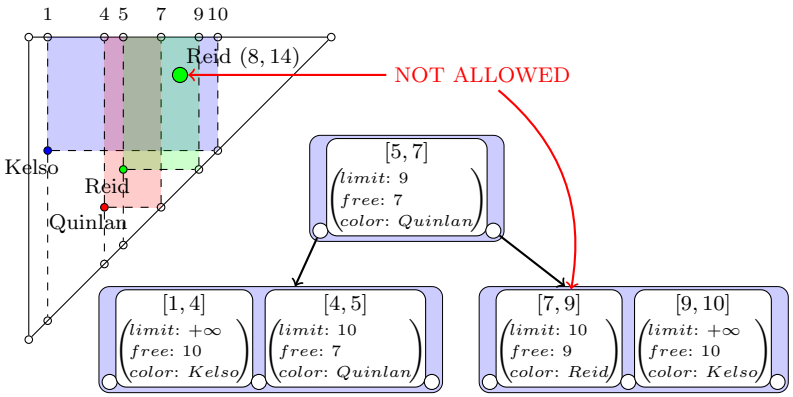

Fig. 22 An example of how insertion fails in the case of $PatId = 2$ of Fig. 1: a tuple with color *Reid* is searched into the B-tree $\mathcal{BT}_V$ using the first coordinate 9 and then the second coordinate 14 is compared with the retrieved node n. As it exceeds the limit $free_n$ with a different color for n (*Kelso* in this case), the tuple violates ITFD $PatId \rightarrow_O Phy$

2. Bettini, C., Jajodia, S.G., Wang, S.X.: Time Granularities in Databases, Data Mining and Temporal Reasoning. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2000)
3. Böhlen, M.H., Jensen, C.S.: Sequenced semantics. In: Liu and Özsu [21], pp. 2619–2621
4. Böhlen, M.H., Jensen, C.S., Snodgrass, R.T.: Nonsequenced semantics. In: Liu and Özsu [21], pp. 1913–1915
5. Chittaro, L., Montanari, A.: Temporal representation and reasoning in artificial intelligence: Issues and approaches. Ann. Math. Artif. Intell. **28**(1-4), 47–106 (2000)
6. Combi, C.: Modeling temporal aspects of visual and textual objects in multimedia databases. In: TIME, pp. 59–68. IEEE Computer Society (2000)
7. Combi, C., Degani, S., Jensen, C.S.: Capturing temporal constraints in temporal ER models. In: Q. Li, S. Spaccapietra, E.S.K. Yu, A. Olivé (eds.) ER, *Lecture Notes in Computer Science*, vol. 5231, pp. 397–411. Springer (2008)
8. Combi, C., Keravnou-Papailiou, E., Shahar, Y.: Temporal Information Systems in Medicine. Springer-Verlag New York, Inc., New York, NY, USA (2010)
9. Combi, C., Montanari, A., Sala, P.: A uniform framework for temporal functional dependencies with multiple granularities. In: D. Pfoser, Y. Tao, K. Mouratidis, M.A. Nascimento, M.F. Mokbel, S. Shekhar, Y. Huang (eds.) SSTD, *Lecture Notes in Computer Science*, vol. 6849, pp. 404–421. Springer (2011)
10. Combi, C., Oliboni, B., Quintarelli, E.: Modeling temporal dimensions of semistructured data. J. Intell. Inf. Syst. **38**(3), 601–644 (2012)
11. Combi, C., Sala, P.: Temporal functional dependencies based on interval relations. In: C. Combi, M. Leucker, F. Wolter (eds.) TIME, pp. 23–30. IEEE (2011)
12. Currim, F., Currim, S., Dyreson, C.E., Snodgrass, R.T., Thomas, S.W., Zhang, R.: Adding temporal constraints to XML Schema. IEEE Trans. Knowl. Data Eng. **24**(8), 1361–1377 (2012)
13. Currim, F., Ram, S.: Conceptually modeling windows and bounds for space and time in database constraints. Commun. ACM **51**(11), 125–129 (2008)
14. Hoffman, K., Mehlhorn, K., Rosenstiehl, P., Tarjan, R.E.: Sorting Jordan sequences in linear time using level-linked search trees. Information and Control **68**(1-3), 170–184 (1986)
15. Huddleston, S., Mehlhorn, K.: Robust balancing in B-trees. In: P. Deussen (ed.) Theoretical Computer Science, *Lecture Notes in Computer Science*, vol. 104, pp. 234–244. Springer (1981)
16. Jensen, C.S., Snodgrass, R.T.: Temporal data models. In: Liu and Özsu [21], pp. 2952–2957
17. Jensen, C.S., Snodgrass, R.T.: Temporal database. In: Liu and Özsu [21], pp. 2957–2960
18. Jensen, C.S., Snodgrass, R.T., Soo, M.D.: Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng. **8**(4), 563–582 (1996)
19. Khatri, V., Snodgrass, R.T., Terenziani, P.: Atelic data. In: Liu and Özsu [21], pp. 142–143
20. Khatri, V., Snodgrass, R.T., Terenziani, P.: Telic distinction in temporal databases. In: Liu and Özsu [21], pp. 2911–2914
21. Liu, L., Özsu, M.T. (eds.): Encyclopedia of Database Systems. Springer US (2009)
22. Montanari, A., Puppis, G., Sala, P.: A decidable spatial logic with cone-shaped cardinal directions. In: E. Grädel, R. Kahle (eds.) CSL, *Lecture Notes in Computer Science*, vol. 5771, pp. 394–408. Springer (2009)
23. Montanari, A., Puppis, G., Sala, P.: Maximal decidable fragments of Halpern and Shoham's modal logic of intervals. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) ICALP (2), *Lecture Notes in Computer Science*, vol. 6199, pp. 345–356. Springer (2010)
24. Shoham, Y.: Temporal logics in AI: Semantical and ontological considerations. Artif. Intell. **33**(1), 89–104 (1987)
25. Snodgrass, R.T. (ed.): The TSQL2 Temporal Query Language. Kluwer (1995)
26. Terenziani, P., Snodgrass, R.T.: Reconciling point-based and interval-based semantics in temporal relational databases: A treatment of the telic/atelic distinction. IEEE Trans. Knowl. Data Eng. **16**(5), 540–551 (2004)
27. Terenziani, P., Snodgrass, R.T., Bottrighi, A., Torchio, M., Molino, G.: Extending temporal databases to deal with telic/atelic medical data. Artificial Intelligence in Medicine **39**(2), 113–126 (2007)
28. Thalheim, B.: Entity-Relationship Modeling - Foundations of Database Technology. Springer (2000)

29. Thalheim, B.: Integrity constraints in (conceptual) database models. In: R. Kaschek, L. Delcambre (eds.) The Evolution of Conceptual Modeling, *Lecture Notes in Computer Science*, vol. 6520, pp. 42–67. Springer Berlin / Heidelberg (2011)
30. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, Volume II. Computer Science Press (1989)
31. Venema, Y.: A modal logic for chopping intervals. J. Log. Comput. **1**(4), 453–476 (1991)
32. Vianu, V.: Dynamic functional dependencies and database aging. J. ACM **34**(1), 28–59 (1987)
33. Wang, X.S., Bettini, C., Brodsky, A., Jajodia, S.: Logical design for temporal databases with multiple granularities. ACM Trans. Database Syst. **22**(2), 115–170 (1997)
34. Wijsen, J.: Design of temporal relational databases based on dynamic and temporal functional dependencies. In: Temporal Databases, pp. 61–76 (1995)
35. Wijsen, J.: Temporal FDs on complex objects. ACM Trans. Database Syst. **24**(1), 127–176 (1999)
36. Wijsen, J.: Temporal dependencies. In: Liu and Özsu [21], pp. 2960–2966
37. Wijsen, J.: Temporal integrity constraints. In: Liu and Özsu [21], pp. 2976–2982