The Markov Reformulation Theorem

Michael Kassoff and **Michael Genesereth**
Logic Group, Department of Computer Science
Stanford University
{mkassoff, genesereth}@cs.stanford.edu

Abstract

In this paper, we define an abstract formal model of dynamic constraints and show that, if the schema is modifiable, the constraints can always be reformulated to Markov constraints.

Introduction

A *dynamic constraint* is a condition that must hold true across two or more states of a database. "Salaries never decrease" and "once a student drops out of the PhD program, he should not be readmitted" are examples of a dynamic constraints.

Previous papers that have studied dynamic constraints have done so from the perspective of a particular specification language, such as variations on Temporal Logic (Lipeck and Zhou 1991; Chomicki 1992; Bidoit and Amo 1995), or Transaction Logic (Bonner, Kifer, and Consens 1993). In this paper, we propose an abstract model of dynamic constraints that is independent of the specification language. The model treats dynamic constraints as sets of valid database histories.

A *Markov* constraint is a dynamic constraint across two consecutive states of a database. Intuitively, a Markov constraint constrains the valid states a database can move to, given only the current state of the database.

It is an open problem whether Markov constraints are expressive enough to model dynamic constraints in general. We propose the Markov Reformulation Theorem proving that this is indeed the case. We give a constructive proof that converts a database with arbitrary dynamic constraints to a database with a Markov constraint, plus an *initial state constraint* that constrains the first state of the database. As part of the construction, we add some new relations to the database schema.

The rest of this paper is organized as follows. We begin by giving an informal classification of the different types of dynamic database constraints. We then make the informal classifications precise. We then move to the major result of this paper, The Markov Reformulation Theorem. Finally, we do a brief analysis of the space complexity of database constraints.

Types of database constraints

In this section, we give an informal overview of the different types of database constraints. In the next section, we make these informal notions precise.

A sequence of consecutive database states is called a *history*. A *static constraint* is a condition that must hold true for each individual state of a database. A *dynamic constraint* is a condition that must hold true across two or more states of a database. Examples of dynamic constraints are "salaries never decrease", and "relation r is append-only."

To help illustrate the different types of dynamic constraints, we turn to the game of chess. We will represent the state of a chess match with a database consisting of a single relation $board$, which represents the current state of the board. Rows are represented by the numbers 1-8, and columns by the letters a-h. Piece types are represented by their names $(pawn, king, queen, rook, knight, bishop)$. Colors are $white$ and $black$. One state is:

$board(a, 1, rook, white)$
$board(b, 8, knight, white)$
$board(e, 1, king, white)$
$board(e, 8, king, black)$

There are a number of constraints on the possible histories of the chess database, corresponding to the rules of the game. For example, each square has at most one piece on it. Since this constraint can be expressed on each individual state, independent of the rest of the database history, it is a *static* constraint. There is also a constraint that states that pieces cannot move onto squares currently holding a piece of the same color. Since this constraint depends only upon two consecutive states of the database, it is known as a *transition* constraint.

A third constraint is the en passant rule of chess, which allows a pawn to capture an opposing pawn that has just moved forward two squares as if the pawn had only moved forward one square, but only during the move immediately following the opposing pawn's move. This can be captured with a constraint that relates three consecutive database states. We call such constraints k-deep dynamic constraints, where in this case k=3.

A final constraint states that a player cannot castle if either the rook or the king involved have previously moved. Since

this constraint can depend on states arbitrarily far in the past, it is a *pure dynamic* constraint.

Formal definitions

We now give formal definitions of the concepts intuitively explained in the previous section.

Database Histories

A *database history* is a sequence of database states. Histories always have a start state and may be finite or countably infinite. For example, $\langle\{p(a,b), r(a)\}, \{p(a,b), q(c,d,e), r(b)\}, \{q(c,d,e)\}\rangle$ is a history of length 3.

The *initial state* of a database history is the first element of the history.

A *subhistory* of a database history is a consecutive subsequence of the history.

A *prefix* of a database history is a subhistory of the history that includes the initial state of the history. An n-ary prefix is a prefix of length n.

A *transition* is a database history of length 2.

Database Constraints

A *static constraint* for a database is a set of states of the database. States that are in the set are considered *valid*, while states that are not in the set are considered *invalid*.

A *dynamic constraint* for a database is a set of database histories for the database. A history h is *valid* with respect to a dynamic constraint c iff h is a prefix of some history in c, otherwise h is considered to be *invalid*. For example, given the dynamic constraint $\{\langle\{p,q\},\{p,r\},\{s\}\rangle, \langle\{q,r\},\{s,t\},\{s\}\rangle\}$, the history $\langle\{p,q\},\{p,r\},\{s\}\rangle$ is valid but $\langle\{p,r\},\{s\}\rangle$ is invalid.

The *dynamic representation* of static constraint c is a dynamic constraint $[\![c]\!]^{static}$ where a history is in $[\![c]\!]^{static}$ iff the states of the history are in c.

A *finite-depth constraint* for a database is a set of sequences of database states of length k, for some integer $k \geq 1$, known as the *depth*. The *dynamic representation* $[\![c]\!]^{depth}$ of a finite-depth constraint c of depth k is a dynamic constraint where a history h is in $[\![c]\!]^{depth}$ iff all n-length subsequences of h are in c.

A *transition constraint* for a database is a finite-depth constraint of depth 2.

An *initial-state constraint* for a database is a static constraint. The *dynamic representation* of an initial-state constraint c is a dynamic constraint $[\![c]\!]^{initial}$ where a history h is in $[\![c]\!]^{initial}$ iff the initial state of h is in c.

The *dynamic representation* of a set of dynamic constraints C is a dynamic constraint $[\![C]\!]^{\cap}$ such that $h \in [\![C]\!]^{\cap}$ iff h is in every $c \in C$.

If a dynamic constraint c is the dynamic representation of a constraint c', we say that c is *equivalent* to c'.

A dynamic constraint c' for a database with schema R' and universe U' *maintains* a dynamic constraint c for a database with schema $R \subseteq R'$ and universe $U \subseteq U'$ iff there is a surjection m from c' to c such that for each history $h' = \langle s'_1, \ldots\rangle \in c'$, $m(h') = \langle s_1, \ldots\rangle$ is a history of the same length as h', such that for each i, $1 \leq i \leq \textsc{Length}(h)$, the restriction of s'_i to R and U equals s_i. A set of dynamic constraints C' *maintains* a set of constraints C iff $[\![C']\!]^{\cap}$ maintains $[\![C]\!]^{\cap}$.

A history is a *valid prefix* of a dynamic constraint iff it is a prefix of some history in the dynamic constraint.

The Markov Reformulation Theorem

Static constraints are the simplest of all the classes of constraints given in the previous section, as they only consider individual database states. Transition constraints are the next simplest, as they consider only consecutive pairs of states. Beyond this, we have constraints that relate 3 consecutive states or more (including countably infinite numbers of states).

The natural question that arises after defining these classes of constraints is, are they all necessary? For example, in our chess example, we formulated the castling rule as a pure dynamic constraint, using only the $board$ relation. However, there is nothing inherent about the castling rule of chess that requires it to be formulated as a pure dynamic constraint. Indeed, we could introduce a new database relation called *piecemoved* that keeps track of whether a piece has moved previously. Given this modified schema, we can reformulate our castling constraint into a transition constraint.

It turns out that, as long as the database schema can be extended with auxiliary relations like $piecemoved$, it is *always* possible to maintain a set of dynamic constraints with a transition constraint and an initial state constraint. We give a proof of this below. This is an important result, as it motivates our choice of Markov Change Logic for expressing database dynamics.

We begin with a few definitions that will allow us to compactly express our theorem. We write $\langle R, U, C\rangle$ for a database with schema R, universe U, and constraints C. If $\langle R, U, C\rangle$ is a database and s' is a state of database $\langle R', U', C'\rangle$ such that $R \subseteq R'$ and $U \subseteq U'$, the *restriction* of s' to R is the database state s such that $r(\bar{t}) \in s$ iff $r(\bar{t}) \in s'$ and $r \in R$. We say that s' *extends* s from R iff s is the restriction of s' to R.

We say that a database $D' = \langle R', U', C'\rangle$ is a *reformulation* of a database $D = \langle R, U, C\rangle$ iff (1) $R \subseteq R'$, (2) $U \subseteq U'$, (3) C' maintains C. A database $D = \langle R, U, C\rangle$ is *Markov* iff C is equivalent to a transition constraint and

an initial state constraint. We say that a database $D' = \langle R', U', C' \rangle$ is a *Markov reformulation* of a database $D = \langle R, U, C \rangle$ iff D' is a reformulation of D and D' is Markov.

We are now ready to state the theorem.

Theorem 1 (Markov Reformulation). *For every database $D = \langle R, U, C \rangle$, there is a database $D' = \langle R', U', C' \rangle$ such that (1) $R \subseteq R'$, (2) $U \subseteq U'$, (3) C' is equivalent to a transition constraint and an initial state constraint, and (4) C' maintains C.*

We will give a constructive proof. Before delving into the proof, we first give an example reformulation that illustrates the construction we will use in the proof.

Example 1. Consider the following database:

Let $R = \{r\}$
Let $U = \{a, b, c, d\}$
Let $C = \{\{ \langle \{r(a)\}, \{r(b)\}, \{r(c)\} \rangle,$
$\langle \{r(a)\}, \{r(b)\}, \{r(d)\} \rangle,$
$\langle \{r(c)\}, \{r(b)\} \rangle \}\}$
Let $D = \langle R, U, C \rangle$

Note that C consists of three histories. The first and the second are of length 3 and the last is of length 2. In each database exactly one fact is true, either $r(a)$, $r(b)$, $r(c)$, or $r(d)$.

We will simulate each history h in C with an initial state constraint and a transition constraint. To do this, we break apart each history into its component transitions, and mark where each transition is applicable by keeping track of (1) how deep the transition appears in the history, using a new counter relation $timestep$, and (2) the previous states of the r relation, kept in a new relation r'.

Let $R' = \{r, r', timestep\}$

Let $U' = \{a, b, c, d, 1, 2, 3\}$

Let $initial = \{\{timestep(1), r(a)\},$
$\{timestep(1), r(c)\}\}$

Let $transition = \{$
$\{\langle \{timestep(1), r(a)\},$
$\{timestep(2), r'(a, 1), r(b)\} \rangle,$
$\{\langle \{timestep(2), r'(a, 1), r(b)\},$
$\{timestep(3), r'(a, 1), r'(b, 2), r(c)\} \rangle,$
$\{\langle \{timestep(2), r'(a, 1), r(b)\},$
$\{timestep(3), r'(a, 1), r'(b, 2), r(d)\} \rangle,$
$\{\langle \{timestep(1), r(c)\},$
$\{timestep(2), r'(c, 1), r(b)\} \rangle\}$

Let $C' = \{[\![initial]\!]^{initial}, [\![transition]\!]^{depth}\}$

Then (1) $R \subseteq R'$, (2) $U \subseteq U'$, (3) C' is equivalent to a transition constraint and an initial state constraint, and (4) C' maintains C. □

We are now ready to give a formal proof of the theorem.

Proof of Theorem 1. We give a constructive proof, by generalizing the construction we used in Example 1. For each relation r in R with arity m, let r' be a new relation with arity $m+1$, and let R^* be the set of these new relations. Let $timestep$ be a new relation with arity 1. Let $\mathbb{N}$ be the set of positive integers.

Let $R' = R \cup R^* \cup \{timestep\}$.

Let $U' = U \cup \mathbb{N}$.

Let $c = [\![C]\!]^{\cap}$.

Given a database state s and a positive integer k, define ARCHIVE$[s, k]$ as $\{r'(\bar{t}, k) \mid r(\bar{t}) \in s\}$. For example, given database state ARCHIVE$[\{r(b), s(c, d)\}, 2] = \{r'(b, 2), s'(c, d, 2)\}$.

For each history $h = \langle s_1, \ldots \rangle \in c$, let $h' = \langle s'_1, \ldots \rangle$ be a history of the same length as h such that each s'_k, $l \leq k \leq$ LENGTH(h), is defined as $\{timestep(k)\} \cup (\bigcup_{i=1}^{k-1}$ ARCHIVE$[s, i]) \cup s_k$. For example, for history $\langle s_1, s_2, s_3 \rangle = \langle \{r(a)\}, \{r(b)\}, \{r(c)\} \rangle$, $s'_3 = \{timestep(3), r'(a, 1), r'(b, 2), r(c)\}$.

Now, let $c' = \{h' \mid h \in c\}$

Let $C' = \{c'\}$

Let $D' = \langle R', U', C' \rangle$.

Defining m as a function from c' to c such that $m(h') = h$, we have that m is a surjection from c' to c such that for each history $h' = \langle s'_1, \ldots \rangle \in c'$, $m(h') = \langle s_1, \ldots \rangle$ is a history of the same length as h', such that for each i, $1 \leq i \leq$ LENGTH(h), the restriction of s'_i to R and U is s_i. Thus c' maintains c, and therefore C' maintains C.

It remains to be shown that C' is equivalent to a set of initial state constraints and a set of transition constraints.

For each $h' \in c'$, let $h'[init] = \{s'_1\}$, and let $h'[trans]$ be the set of transitions of h'.

Let $I' = \bigcup_{h' \in c'} h'[init]$. Let $T' = \bigcup_{h' \in c'} h'[trans]$.
Let $C^* = \{[\![I']\!]^{initial}, [\![T']\!]^{depth}\}$

Let $c^* = [\![C^*]\!]^{\cap}$

We claim that $c^* = c'$. We first show that $c' \subseteq c^*$. Say that $h' \in c'$. Since every transition of h' is in $h'[trans]$, and since the start state of h' is in $h'[init]$, we see that

$h' \in c^*$ as well. Thus $c' \subseteq c^*$. We next show that $c^* \subseteq c'$. Say that $h^* \in c^*$. Say for the purpose of contradiction that $h^* \notin c'$. Then there must be some finite prefix of h^* that is not a prefix of some history in c'. Let $p^* = \langle p_1^*, \ldots, p_k^* \rangle$ be the shortest prefix of h^* such that p^* is not a prefix of some history in c'. If p^* is of length 1, then $p_1^* \in I'$ and thus p_1^* is the initial state of some history in c', and p^* is a prefix of some history in c', a contradiction. Thus p^* is of length at least 2. Since p^* is the shortest prefix of h^* such that p^* is not a prefix of some history in c', it must be that $p^- = \langle p_1^*, \ldots, p_{k-1}^* \rangle$ is a valid prefix of some $h' \in c'$. Furthermore, there must be a transition $\langle p_{k-1}^*, p_k^* \rangle \in T'$. But then $p_k^* = h'_k \cup \{timestep(k)\} \cup (\bigcup_{i=1}^{k-1} \textsc{Archive}[h', i])$. Thus p^* is a prefix of h', and we have a contradiction. Thus $h^* \in c'$, and so $c^* \subseteq c'$. Since $c^* \subseteq c'$ and $c' \subseteq c^*$, $c^* = c'$. Thus $[\![C^*]\!]^\cap = [\![C']\!]^\cap$. Since C^* is equivalent to a set of initial state constraints and a set of transition constraints, so is C'.

Thus $D' = \langle R', U', C' \rangle$ is a database with constraints such that (1) $R \subseteq R'$, (2) $U \subseteq U'$, (3) C' is equivalent to a transition constraint and an initial state constraint, and (4) C' maintains C. Since D was arbitrary, we have proved the claim. □

Space Complexity

In the previous section we showed how to reformulate a database with arbitrary dynamic constraints into a database with a transition constraint and an initial state constraint. This is done by adding additional relations to the database to simulate the transition constraints with depth 3 and greater. Those relations, of course, must be stored and take up additional space. In this section, we explore just how much space is required for a Markov reformulation.

We begin with a few definitions.

- Let database D' be reformulation of database $D = \langle R, U, C \rangle$. A history $h' = \langle s'_1, \ldots, \rangle$ for D' and a history $h = \langle s_1, \ldots, \rangle$ *correspond* iff length(h) = length(h') and for each i, $1 \leq i \leq length(h)$, the restriction of s'_i to R is s_i.
- An *encoding* E of a universe U is an injective function that maps each entity $a \in U$ to a binary number b. We assume encodings of natural numbers are monotonic, i.e. if m and n are natural numbers such that $m < n$, $size(E(m)) \leq size(E(n))$.
- The *size* of an entity a relative to an encoding E is equal to the number of digits in $E(a)$.
- The *size* of a datum $p(t_1, \ldots, t_k)$ is $\Sigma_{i=1}^k size(t_i)$.
- The *size* of a database instance $\{p_1, \ldots, p_n\}$ is $\Sigma_{i=1}^n size(p_i)$.
- The *maximum size* of a finite database history $\langle s_1, \ldots, s_k \rangle$ is $max_{i=1}^k size(s_i)$.
- We say that a reformulation D' of a database D is *bounded* iff for each k-ary prefix p of a valid history h of D, if p' is the k-ary prefix of the corresponding history h' of D', then for some number B, $maxsize(p')/maxsize(p) \leq B$.
- We say that a reformulation D' of a database D is *constantly bounded* iff for each k-ary prefix p of a valid history h of D, if p' is the k-ary prefix of the corresponding history h' of D', then for some number B, $maxsize(p') - maxsize(p) \leq B$.
- If every database having a constraint set C has a (constantly) bounded reformulation, then we say that C has a *(constantly) bounded reformulation*.

We now give examples of unbounded, bounded, and constantly bounded reformulations, respectively.

Unbounded reformulations. By Theorem 1, we know that every database D has a Markov reformulation D'. However, in general, using the technique in the proof to construct the Markov reformulation leads to reformulations that are unbounded. This is because the construction in the proof materializes the history of the database up to each state s_i in the s_i. In some cases, an unbounded reformulation is the best we can do. Consider for example a database with a single constraint that forbids the same state from ever occurring twice. In this case, no bounded reformulation will do, because any bounded reformulation can only distinguish between some finite number of past histories.

Bounded reformulations. Conversely, every constraint set consisting of a single finite-depth constraint of depth k has a bounded Markov reformulation. In particular, consider the reformulation of Theorem 1, but modified so that (1) only the previous $k-1$ states are recorded in the r' relations, and (2) the transition constraint depends on the previous $k-1$ states instead of the entire history. This construction works since finite depth constraints of depth k only constrain sets of consecutive of k states. Indeed, any finite set of finite-depth constraints has a bounded Markov reformulation, since any finite set of finite-depth constraints is equivalent to a single finite depth constraint.

Constantly bounded reformulations. An *eventually* constraint for fact $p(\bar{t})$ is a constraint consisting of all histories such that $p(\bar{t})$ is true in some state of the history. It is easy to see that every eventually constraint has a constantly bounded Markov reformulation, where the additional state that is kept is a single datum $occurred$ iff the fact was true previously in the history, or $occurred(no)$ iff not.

Conclusion

In this paper, we defined an abstract semantic model of dynamic constraints that is independent of any particular language used to define dynamic constraints. The abstract model of dynamic constraints as sets of histories can be thought of as a universal semantics for dynamic constraints that can be used to analyze various other dynamic constraint

languages that have been proposed. There are not yet any standard ways of defining dynamic constraints, and it seems that each researcher has defined his own language for doing so. For example, Lipeck uses Temporal Logic to define dynamic integrity constraints (Lipeck and Zhou 1991; Lipeck, Gertz, and Saake 1994; Gertz and Lipeck 1996) Chomicki uses Past Temporal Logic (Chomicki 1992; 1995; Chomicki and Niwiński 1995), and Bidoit uses Linear Temporal Logic (Bidoit and Amo 1995). Transaction Logic (Bonner, Kifer, and Consens 1993) is yet another alternative. See (Silva 1997) for a survey the various approaches to dynamic integrity constraints. It is also worth noting that the Markov Reformulation Theorem is an abstract theorem and may not apply to any particular dynamic constraint language in that the reformulated constraints may not be expressible in that language. Investigating whether particular dynamic constraint languages are closed under reformulation is an interesting route of future study. Analyzing the space complexity of particular dynamic constraint languages is another area that requires future research.

References

Bidoit, N., and Amo, S. D. 1995. A first step towards implementing dynamic algebraic dependencies. In *Proceedings of the 5th International Conference on Database Theory*, ICDT '95, 308–321. London, UK: Springer-Verlag.

Bonner, A.; Kifer, M.; and Consens, M. 1993. Database programming in transaction logic. In *Workshop on Database Programming Languages*, 309–337.

Chomicki, J., and Niwiński, D. 1995. On the feasibility of checking temporal integrity constraints. *J. Comput. Syst. Sci.* 51:523–535.

Chomicki, J. 1992. History-less checking of dynamic integrity constraints. In *Proceedings of the Eighth International Conference on Data Engineering*, 557–564. Washington, DC, USA: IEEE Computer Society.

Chomicki, J. 1995. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.* 20:149–186.

Gertz, M., and Lipeck, U. W. 1996. Deriving optimized integrity monitoring triggers from dynamic integrity constraints. *Data Knowl. Eng.* 20:163–193.

Lipeck, U. W., and Zhou, H. 1991. Monitoring dynamic integrity constraints on finite state sequences and existence intervals. In *FMLDO*, 115–130.

Lipeck, U. W.; Gertz, M.; and Saake, G. 1994. Transitional monitoring of dynamic integrity constraints. *IEEE Data Eng. Bull.* 17(2):38–42.

Silva, M. A. P. e. 1997. Dynamic integrity constraints definition and enforcement in databases: A classification framework. In *Proceedings of the IFIP TC11 Working Group 11.5, First Working Conference on Integrity and Internal Control in Information Systems: Increasing the confidence in Information Systems*, 65–87. London, UK, UK: Chapman & Hall, Ltd.