TECHNICAL REPORT NO. 415

A Data Model for Audio-Video Data

Munish Gandhi * Edward L. Robertson*

gandhim@cs.indiana.edu edrbtsn@cs.indiana.edu

August 24, 1994

*Computer Science Department, Indiana University, Bloomington, IN-47405.

Contents

Abstract

Audio and video data have become critical components of information systems and these systems need to efficiently manage the large storage requirements of this type of data. However, there are no formal data models for audio video data. In this paper, we present an algebraic formalism which attempts to provide the underpinnings for the design and implementation of systems which organize and query such data.

The central class of interest in the formalism is the track, which is both the major repository of structure and the unit of physical storage. Primitive track operations involve moving information of a single track, pointwise combination of two tracks, or manipulation of one track based on the structure of the another. Interrelations between primitives are explored and common user-level operations, such as splicing and overlaying, are defined in terms of the primitives.

1 Introduction

Continuous media (cm), such as audio and video, are becoming increasingly significant data types in hypermedia systems. A key cm property is that it is a sequence of finer grained elements. The size of these elements is usually comparable to a hypermedia node, but the size of the cm is orders of magnitude larger than the size of a node. Hypermedia systems (correctly) abstract away these difference between cm nodes and other nodes for maintaining conceptual simplicity. However, as we will see in the next paragraph, these differences become decisive at the storage level.

Since cm data is large and considered atomic from the hypermedia viewpoint, it is usually stored as a file. A file-oriented organization is unable to take advantage of any possible redundancy. To illustrate, if an audio file has been derived using segments from other audio files, it would be more efficient to store only the derivation specification. Furthermore, in a client-server setting, if access is needed to only a segment of a file, the whole file is unnecessarily ported to the client.

The above problems are symptomatic of a lack of a data model [TL82] which captures and manipulates the structure of continuous media. In this paper, we propose an algebraic data model as a formal basis for the management of continuous media. A system which uses this formalism is conceptualized as having a three layered architecture. These layers are distinguished by the level of granularity at which media data is manipulated.

The algebra focuses on the central layer – the *media track* layer. Since the algebra needs to be independent of specific audio video data formats, a media track is assumed to be a sequence of containers of *uninterpretable* data called *media atoms*. The interpretation of media atoms is delegated to specific media format processors. The algebra also delegates responsibility of

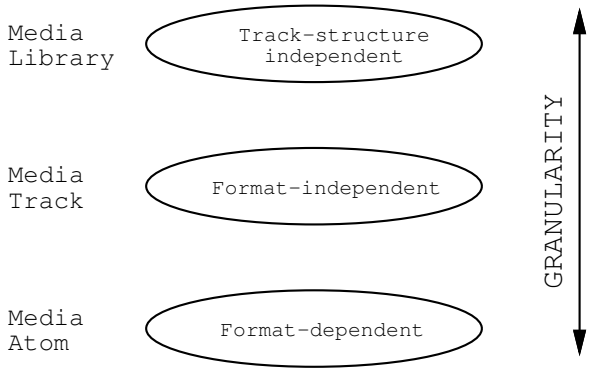


Figure 1: Conceptual architecture

maintaining links between tracks and other miscellaneous data about tracks to the *media library*. The media library, in turn, views a track as being essentially atomic.

While there is work at the extremes of the granularity spectrum, issues at the track level of granularity have been largely unexplored (save for [BGT92, GBT92]). For example, at the finest level of granularity, there exist many multimedia standards for interpreting bit streams as audio or video data [LG91, Wal91, Poh92]. At the coarsest level of granularity, current database technology [EN89] or other evolving paradigms [HBvR94, NKN91] may be used to organize a set of tracks.

In the next section (section 2), we discuss the structure of the algebraic domain. This will be followed by a discussion of the algebraic operators in section 3. Some of these operators have properties similar to the boolean algebra and these will be considered in section 4. We will close by giving a concise definition of the algebra in section 5 and a few concluding remarks in section 6.

2 Algebra: Domain

A basic assumption of the data model is that media streams may be characterized by a sequence of finer grained elements. These finer grained elements are stored as *track atoms* and represented in the model using *track atom identifiers (tad)*. A media stream is represented by a *track* which positions tads along an integer axis. We discuss these concepts in greater detail below.

2.1 Track Atom, TAD and Track

One of the objectives in developing this model was that it should be a natural abstraction for media data. Thereto, we assume media streams to be a sequence of abstract data elements called the *track atoms*. Track atoms have a finer structure but that structure is interpretable by specific media format processors. To get an intuitive feel for track atoms, consider two digitial video formats – MPEG (Moving Pictures Experts Group) [LG91] and JPEG (Joint Photographic Experts Group) [Wal91]. JPEG video is a simple sequence of compressed frames. In this case, a compressed frame is appropriate as a track atom for JPEG streams. MPEG video, on the other hand, compresses (approximately 10) consecutive frames into a unit. These units are appropriate as track atoms for MPEG streams, but since a frame occurs at a granularity level discernible only by an MPEG processor, a frame is not.

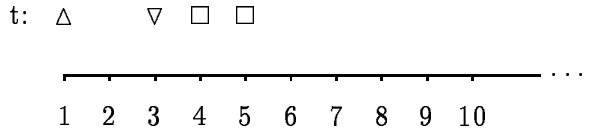
The abstraction objective has another consequence. Since the atoms do not permit any modifications, all manipulations may be done using surrogates. Thus, each track atom is uniquely represented by *Track Atom iDentifiers (tads)* and all operations in the model are done using tads.

A *track* is a partial function from the positive integers $\mathcal{N}$ to the set of tads $\mathcal{TAD}$. This defines the sequence of track atoms constituting the track

by placing the tads representing the atoms on a one dimensional grid of possible positions. For example, a second of video at 30 frames/second (fps) in the JPEG format may be represented by placing the tad of the first frame at the 1st position, the tad of the second at the 2nd position, ..., and the tad of the 30th frame at the 30th position. Similarly, a second of 30 fps and 10 frames/unit of MPEG video may be represented by placing the three tads at positions 1, 11 and 21. Note that the MPEG video representation has 'holes' in it. Since the positions where the track has holes may be represented by a partial function which is undefined at those positions, we chose to represent a track as a function rather than a list.

The above definition of a track does not assume that there is a relationship between the track domain and time. The position grid may be representative of any domain with a starting point and a linear order. For instance, the position grid may be interpreted as bar numbers in a score or as scene numbers in a movie.

We will use diagrams such as the one below to depict tracks. We depict a position grid using a horizontal line marked with position numbers, tads using distinctly shaped icons, and the placement of tads using icons at corresponding positions. Thus, track t is defined at positions 1, 3, 4, and 5, and the tads at positions 4 and 5 refer to the same atom.



2.2 Index Track

The grid positions at which track atoms are placed may be considered as an index into the track. However, tracks may need to be indexed in secondary ways. For example, a video track may be indexed by frame numbers, by time and by scene numbers. In order to accomodate such scenarios, we consider tracks where the placed atoms are the positive integers.

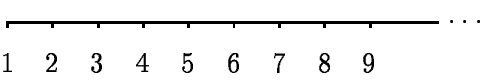
Specifically, an *index track* is a partial map from $\mathcal{N}$ into itself. Since the operations (in section 3) defined on tracks are also applicable for the index tracks, an index track may be considered a track. We formalize this by letting $\mathcal{N} \subset \mathcal{TAD}$ thus making the index tracks a proper subset of the tracks.

As an example, consider a movie track in which each shot is a track atom, and successive positions have successive shots. Since shots in movies are not of fixed length, the grid positions are not indicative of time in the movie. Therefore, we may create an index track which specifies the total duration of the past shots for each shot in the track of shots.

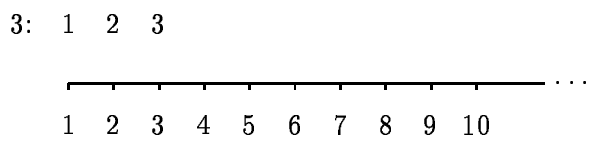
The model also employs index tracks to code the boolean constants and the natural numbers. Consider the boolean constants first. The index track $TRUE$ ($\stackrel{\Delta}{=} \{\langle 1, 1\rangle\}$) will be used to signify the boolean 'true'. The track $FALSE$ ($\stackrel{\Delta}{=} \{\}$) will be used to signify the boolean 'false'. Given these *boolean tracks* as representations of the boolean constants, we will code the boolean operations using the primitive track operations in section 3.

$TRUE$: 1

$FALSE$:



We represent a natural number n by an index track with all positions until n occupied by the respective position numbers, that is, by the track $\{\langle i, i\rangle | i \leq n\}$. Thus, 3 would be represented by the *integer track*:



Similar to the boolean algebra, we will code integer addition, subtraction, multiplication, and division using the track operations in section 3. Moreover, we will use both the integer and boolean tracks to code integer comparisons.

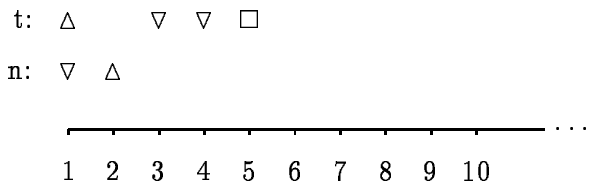
3 Algebra: Operations

3.1 Track Transposition

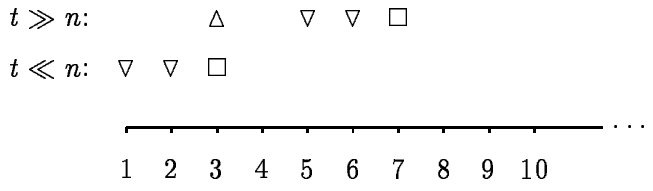
Transposition operations change the placement of the atoms in the position grid without affecting the relative order of the atoms. These operations are fundamental to meeting track alignment requirements. For example, starting a movie after an advertisement would require the movie track be shifted right by a magnitude equal to the duration of the advertisement track. Other alignments are also possible. Consider a 30 fps, 10 frames/unit, MPEG track with consecutive units occupying consecutive positions, and a 30 fps JPEG track with consecutive frames occupying consecutive positions. Stretching the MPEG track by a factor of 10 would align the timing of both tracks, that is, the ith position in both tracks could be interpreted as representing the i/30th second of the track. We call such tracks, that is, tracks where a position has the same interpretation in either track, as

associated tracks.

All transpositions are binary operations. The first operand specifies the track which is being transposed, and the second operand specifies the magnitude of the transposition. More precisely, the position at which the last tad is placed in the second operand defines the magnitude. If the second operand is empty the result of the transposition is an empty track. We will illustrate the transpositions using t as the first operand and n as the second operand. Note that track n specifies the size of transposition as 2.

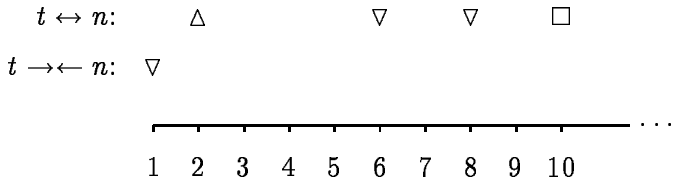


The *shift right* (*shift left*) operation $\gg$ ($\ll$) shifts each atom to the right (left) by the specified magnitude. The shift left operation, however, may lose atoms which fall left of the 1st position after shifting. Thus, one may not be able to recover the original track after a shift left operation on the track. In this sense, the shift left is a 'lossy' operation.



The *stretch* operation $\leftrightarrow$ places each atom at a position m times its original position, where m is the specified magnitude of the transposition. The *shrink* operation $\rightarrow\leftarrow$ places an atom at a position only if the original position of that atom was m times the new position, where m is the magnitude

of the transposition. Since all the atoms placed in positions which are not integral multiples of m are dropped in this operation, the shrink operation is lossy too.



The transposition operations are summarized in the table below. We assume $i, j, k \in \mathcal{N}$; $o, o', o'' \in \mathcal{TAD}$; $n, t \in \mathcal{T}$; and, unless specified otherwise, these variable are existentially bound. For readability, we define a macro $LAST(j, n)$ to represent the last position j in track n where $n(j)$ is defined. If n is undefined at all points, j is 0.

$$LAST(j,n) = \left(\begin{array}{l} (\langle j, o' \rangle \in n, \nexists o'' : k > j, \langle k, o'' \rangle \in n), \text{ or} \\ (\nexists o' : \langle i.o' \rangle \in n, j = 0) \end{array} \right)$$

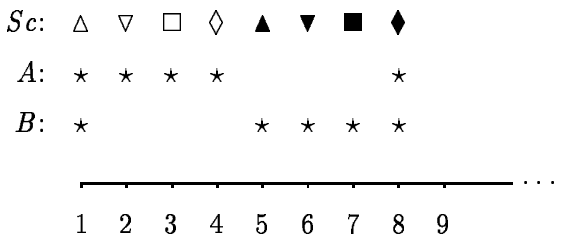
3.2 Pointwise Splicing

The pointwise splice operations provide basic capabilities to compare and combine two tracks. We introduce these operations by considering possible queries that may arise given a scene where two characters, say A and B, are having a conversation. Assume our database has three tracks. The track Sc is the scene from the movie; the associated track A represents those parts of the scene where A is in the frame; and the track B those parts where B is in the frame.

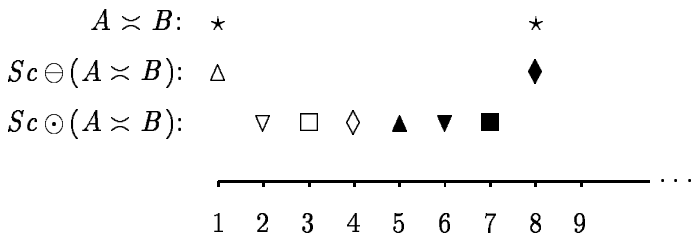
Let's start with "Give the frames where both characters are present". The common operation $\asymp$, which creates a track with a tad at only those

Operation	Signature	Definition
Shift right	$\gg: \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \gg n \quad \triangleq \quad \left\{ \langle i+j, o\rangle \mid \begin{array}{l} \langle i, o\rangle \in t, \\ LAST(j, n) \end{array} \right\}$
Shift left	$\ll: \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \ll n \quad \triangleq \quad \left\{ \langle i, o\rangle \mid \begin{array}{l} \langle i+j, o\rangle \in t, \\ LAST(j, n) \end{array} \right\}$
Expand	$\leftrightarrow: \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \leftrightarrow n \quad \triangleq \quad \left\{ \langle i*j, o\rangle \mid \begin{array}{l} \langle i, o\rangle \in t, i*j > 0 \\ LAST(j, n) \end{array} \right\}$
Shrink	$\rightarrow\leftarrow: \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \rightarrow\leftarrow n \quad \triangleq \quad \left\{ \langle i, o\rangle \mid \begin{array}{l} \langle i*j, o\rangle \in t, \\ LAST(j, n) \end{array} \right\}$

Table 1: Transposition operations



positions where the tads of the operands are equal, may be used to answer this query $(A \asymp B)$. Further, one may "extract those parts of the scene with both characters" using the *mask* operation $\ominus$ which selects those segments of the first track where the second track is defined $(Sc \ominus (A \asymp B))$. A similar operation, *Revmask* $\odot$, which selects those segments of the first track where the second track is *not* defined, may be used to "extract the frames where only one character is present" $(Sc \odot (A \asymp B))$.



The *xorsplice* and the *overlay* operators may be used to combine tracks. The track created by an xorsplice operation $\oplus$ has a tad at a certain position only if that tad occurs in either of the operand tracks and there is no tad in the other track at that position. Thus one may "create the annotation track where only one of the characters is present" using xorsplice $(A \oplus B)$. The *splice* operation $\uplus$ copies the first track operand into the resultant track, and, fills the empty positions by tads at those positions from the second track operand. This is similar to the xorsplice operation but for the tads from the first track at positions where both the tracks have tads. This difference makes overlay non-commutative while xorsplice is commutative. One may overlay tracks to "create the annotation track where either of the characters is present" $(A \uplus B)$.

All the operations in this section are not primitive. Section 4 defines a family of pointwise splices operations and shows that any pointwise splice

may be obtained from $\oplus$ and $\asymp$ operators. Since the remaining operators in this section ($\uplus$, $\ominus$ and $\odot$) are pointwise splices, they may be derived from $\oplus$ and $\asymp$ operators.

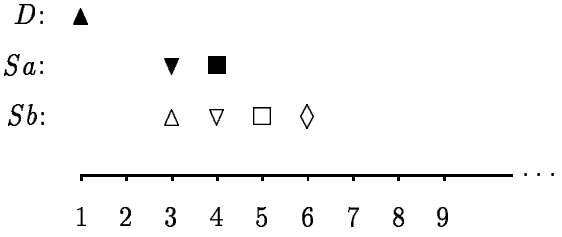
The pointwise splice operations may be summarized in the table below. In this table, we assume $i \in \mathcal{N}$; $o, o' \in \mathcal{TAD}$; $t, t' \in \mathcal{T}$; and, unless specified otherwise, these variable are existentially bound.

Operation	Signature	Definition
Common	$\asymp : \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \asymp t' \quad \triangleq \quad \left\{ \langle i, o \rangle \mid \begin{array}{l} \langle i, o \rangle \in t, \\ \langle i, o \rangle \in t' \end{array} \right\}$
Xorsplice	$\oplus : \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \oplus t' \quad \triangleq \quad \left\{ \langle i, o \rangle \mid \begin{array}{l} \langle i, o \rangle \in t, \nexists o', \langle i, o' \rangle \in t', \text{or} \\ \langle i, o \rangle \in t', \nexists o', \langle i, o' \rangle \in t \end{array} \right\}$
Overlay	$\uplus : \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \uplus t' \quad \triangleq \quad \left\{ \langle i, o \rangle \mid \begin{array}{l} \langle i, o \rangle \in t, \text{or} \\ \langle i, o \rangle \in t', \nexists o', \langle i, o' \rangle \in t, \end{array} \right\}$
Mask	$\ominus : \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \ominus t' \quad \triangleq \quad \left\{ \langle i, o \rangle \mid \begin{array}{l} \langle i, o \rangle \in t, \\ \langle i, o' \rangle \in t', \end{array} \right\}$
Revmask	$\odot : \quad \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T}$	$t \odot t' \quad \triangleq \quad \left\{ \langle i, o \rangle \mid \begin{array}{l} \langle i, o \rangle \in t, \\ \nexists o', \langle i, o' \rangle \in t', \end{array} \right\}$

Table 2: Pointwise splices

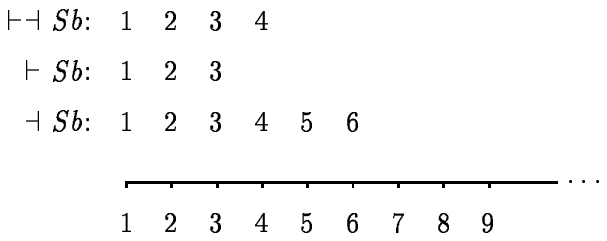
3.3 Domain information operations

The domain information operations on a track return information on the positions where the track is defined. As before, we motivate these operations using an example. Consider a disk jockey who has to decide the last four minutes of programming for her show. She has two choices – a dedication D for a song and the song Sa, or another song Sb (the songs are presumably extracted from other tracks using mechanisms outlined in section 3.2). We assume the track atoms are one minute audio segments.



A good starting point for the jockey is to know the lengths of the different tracks. This may be determined by the unary operator *extent* $\vdash\dashv$ which returns an integer track representing the length of the operand track ($\vdash\dashv Sb$ is shown below). She notices that the length of Sb matches her requirement exactly and proceeds to end her show with it. However, since Sb does not begin at 1, she must shift left Sb by a magnitude equal to the starting position of the track (more precisely, the magnitude minus one because tracks start at position 1). The *head* ($\vdash$) of a track, which returns an integer track which represents the position of the first atom in the track, may be used to specify this magnitude ($\vdash\ Sb$). An analogous operator *tail* ($\dashv$), returns an integer track representing the position of the last atom in the track ($\dashv Sb$).

These operations are formally specified below. We assume $i, j \in \mathcal{N}$;



$o, o' \in \mathcal{TAD}$; $t \in \mathcal{T}$; and, unless specified otherwise, these variable are existentially bound. For readability, we define macros $FIRST(j,t)$ to represent the first position j in track t where $t(j)$ is defined, and $LAST(j,t)$ to represent the last position j in track t where $t(j)$ is defined. If t is undefined at all points, j is 0.

$$FIRST(j,t) = \left(\begin{array}{l} (\langle j, o' \rangle \in t, \nexists o'' : k < j, \langle k, o'' \rangle \in n), \text{ or} \\ (\nexists o' : \langle i.o' \rangle \in t, j = 0) \end{array} \right)$$

$$LAST(j,t) = \left(\begin{array}{l} (\langle j, o' \rangle \in t, \nexists o'' : k > j, \langle k, o'' \rangle \in n), \text{ or} \\ (\nexists o' : \langle i.o' \rangle \in t, j = 0) \end{array} \right)$$

Operation	Signature	Definition
Extent	$\vdash\dashv : \quad \mathcal{T} \mapsto \mathcal{T}$	$\vdash\dashv t \quad \triangleq \quad \left\{ \langle i,i \rangle \mid \begin{array}{l} i \leq k-j+1, j>0 \\ FIRST(j,t), LAST(k,t) \end{array} \right\}$
Head	$\vdash : \quad \mathcal{T} \mapsto \mathcal{T}$	$\vdash t \quad \triangleq \quad \left\{ \langle i,i \rangle \mid \begin{array}{l} i \leq j, \\ FIRST(j,t) \end{array} \right\}$
Tail	$\dashv : \quad \mathcal{T} \mapsto \mathcal{T}$	$\dashv t \quad \triangleq \quad \left\{ \langle i,i \rangle \mid \begin{array}{l} i \leq j, \\ LAST(j,t) \end{array} \right\}$

Table 3: Domain Information

3.4 Boolean algebra and integer manipulations

As one may have guessed, the pointwise splices are sufficient to simulate boolean algebra. We define the boolean 'and', 'or' and 'not' in terms of the pointwise splices in table 4. Here, $t, t' \in \{TRUE\ , FALSE\ \}$.

Operation	Symbol	Definition
Not	$\neg$	$\neg t \stackrel{\Delta}{=} t \oplus TRUE$
Or	$\vee$	$t \vee t' \stackrel{\Delta}{=} (t \oplus t') \oplus (t \asymp t')$
And	$\wedge$	$t \wedge t' \stackrel{\Delta}{=} (t \asymp t')$

Table 4: Boolean operations

The track algebra has sufficent power to perform the integer operations 'addition', 'subtraction', 'multiplication', and 'division'. The disc jockey may use such a functionality to determine, say, the total length of the first possibility which consisted of the dedication and the song $((\vdash\dashv D) + (\vdash\dashv Sa))$. Table 5 gives the expressions for these operators in terms of the operators that we have already discussed.

Operation	Symbol	Definition
Addition	$+$	$i + j \stackrel{\Delta}{=} \dashv((i \gg j) \oplus (j \gg i))$
Subtraction	$-$	$i - j \stackrel{\Delta}{=} \dashv((i \gg j) \oplus (j \gg i))$
Multiplication	$*$	$i * j \stackrel{\Delta}{=} \dashv(i \leftrightarrow j)$
Division	$/$	$i/j \stackrel{\Delta}{=} \dashv(i \rightarrow\!\leftarrow j)$

Table 5: Integer operations

The track operators are also capable of handling integer comparisons 'greater than', 'less than', and 'equal to'. Depending on whether the comparison holds or not, the operations result in the tracks $TRUE$ or $FALSE$, respectively. Table 6 below defines these derived operators. Going back to

our example, if the disc jockey could finish her show slightly earlier and preferred the first choice, she may use the comparison operators to check if its duration was indeed lesser than the time contraint $(((\vdash \dashv D)+(\vdash \dashv Sa)) < 4)$.

Operation	Symbol	Definition
Less than	$<$	$i < j \triangleq TRUE \ominus ((j \odot i) \ll (\vdash (j \odot i) \gg 1))$
Equal to	$=$	$i = j \triangleq TRUE \ominus (i \oplus j) \ll (\vdash (i \oplus j) \gg 1))$
Greater than	$>$	$i > j \triangleq TRUE \ominus ((i \odot j) \ll (\vdash (i \odot j) \gg 1))$

Table 6: Integer comparisons

Since we have boolean algebra capabilities (table 4), they may be used to define complex conditions on the tracks. For example, $(i \leq j)$ is simply $((i = j) \wedge (i < j))$, and the disc jockey can ensure that the last segment L is between three minutes and five minutes long by using the expression $(L \geq 3) \wedge (L \leq 5)$. In fact, the algebra is sufficiently powerful to represent any of the 13 mutually exclusive temporal relationships between two intervals [All83] ('Before', 'Equals', 'Overlaps' and 'Contains' are a few of these). For example, if I and J are two tracks, then $I\ Contains\ J$ is the expression $(\vdash I \leq \vdash J) \wedge (\dashv I \geq \dashv J)$.

4 Family of pointwise splices

The operators in section 3.2 are only five of the operations in a family of pointwise splice operations. We define and analyze this family here.

Imagine a track splicing machine which produces, in one pass, an output track given two input tracks. The content at each position of the output track is restricted to being the content of either of the input tracks at that position or being undefined. The choice between these possibilities is made using *splice rules*.

What are reasonable criterion that one may use to define splice rules? The resultant track in the overlay operation is based on whether the input tracks have tads in that position; while the resultant track in the common operation is based on whether the tads in the two tracks are equal or not. Thus, we consider the splice rules which use the definedness and equality criterion for their specification. Given these criterion, the corresponding positions on the two input tracks will exhibit either of the following patterns:

1. A tad is defined for both tracks, but the tads are not equal. In this case, the resultant tad may either be from the first or the second track or be undefined.

2. A tad is defined for both tracks and the tads are equal. In this case, the resultant tad may either be the tad defined on both tracks or be undefined.

3. The first track has a tad, but no tad is defined for the second track. Here, the resultant tad may either be the tad from the first track or be undefined.

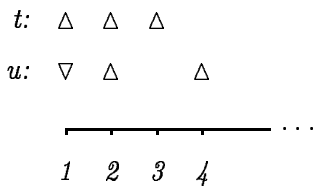
4. The second track has a tad, but no tad is defined for the first track. Here, the resultant tad may either be the tad from the second track or be undefined.

5. Tads are not defined for either of the tracks. Here, the resultant tad will be undefined.

Since the last pattern always results in an undefined result, we need only the first four patterns to define a splice rule, and therefore define a pointwise splice.

Definition 4.1 *A splice rule ρ is a track such that $\rho(1) \in \{\Delta, \nabla, \perp\}$, $\rho(i) \in \{\Delta, \perp\}$, $2 \leq i \leq 4$ and $\rho(i) = \perp$, $i > 4$.*

Definition 4.2 *Let Θ be a pointwise splice operator and ρ a splice rule. Then, Θ is the pointwise splice specified by ρ iff $(t \Theta u) = \rho$, where t and u are the tracks below. The undefined ($\perp$) positions are left blank in the figure.*



Notation 4.3 *We will denote a splice operator specified by the splice rule ρ as $\Theta_{(\rho(1),\rho(2),\rho(3),\rho(4))}$.*

Using this notation, we may recognize the common, xorsplice, overlay, mask and revmask operators in table 2 as $\Theta_{(\perp,\Delta,\perp,\perp)}$, $\Theta_{(\perp,\perp,\Delta,\Delta)}$, $\Theta_{(\Delta,\Delta,\Delta,\Delta)}$ $\Theta_{(\Delta,\Delta,\perp,\perp)}$, and $\Theta_{(\perp,\perp,\Delta,\perp)}$, respectively.

Definition 4.4 *We say a set of pointwise splice operators, $\mathcal{S}$, is splice-complete if all the pointwise splices may be expressed in terms of the operators in $\mathcal{S}$.*

Theorem 4.5 *$\{\oplus, \asymp\}$ is splice-complete.*

Proof: Let t and u be the tracks in definition 4.2 and let $t_{(n,a)}$ denote the track which has tad a defined at position n but is undefined at all other positions. Thus, $t_{(n,\perp)}$ denotes the track which is undefined at all points. For $1 \leq n \leq 4$, the expression $t \oplus t$ corresponds to track $t_{(n,\perp)}$. The expressions $((t \asymp u) \oplus t) \oplus ((t \oplus u) \asymp t)$ and $((u \asymp t) \oplus u) \oplus ((u \oplus t) \asymp u)$ correspond to

$t_{(1,\Delta)}$ and $t_{(1,\nabla)}$, respectively. Finally, the expressions $t \asymp u$, $(t \oplus u) \asymp t$, and $(u \oplus t) \asymp u$ correspond to the tracks $t_{(2,\Delta)}$, $t_{(3,\Delta)}$, and $t_{(4,\Delta)}$, respectively.

Now, the expression for the pointwise splice operator $\Theta_{(w,x,y,z)}$ is $t_{(1,w)} \oplus t_{(2,x)} \oplus t_{(3,y)} \oplus t_{(4,z)}$. ■

Lemma 4.6 *A set containing only the pointwise splice $\Theta_{(w,\perp,\Delta,\Delta)}$, where $w \in \{\Delta, \nabla, \perp\}$, is not splice-complete.*

Proof: Let t and u be the tracks in definition 4.2. We claim that property $P(\tau)$ holds for all tracks τ generated from t and u using the operator $\Theta_{(w,\perp,\Delta,\Delta)}$, where $P(\tau)$ is: $\tau(2)$, $\tau(3)$ *and* $\tau(4)$ *contain an odd number of* $\perp$.

The basis holds since both t and u have one $\perp$ in these positions. Let τ be $(e\ \Theta_{(w,\perp,\Delta,\Delta)}\ f)$, where both $P(e)$ and $P(f)$ hold. There are three cases here:

- Both e and f have three $\perp$ in positions 2-4.

  Then, τ has $\perp$ in positions 2, 3, and 4, and $P(\tau)$ holds.

- Only one of e and f has three $\perp$ in positions 2-4.

  Then the other has only one $\perp$ at position p. Now, τ has an $\perp$ only at p. Thus, $P(\tau)$ holds.

- Both e and f have exactly one $\perp$ in positions 2-4.

  Then, either the position of $\perp$s coincide and τ has an $\perp$ at all the positions 2, 3, and 4. Or, the positions of $\perp$ do not coincide and τ has an $\perp$ at only the position where there is no $\perp$ in the original tracks. Thus, $P(\tau)$ holds.

■

Lemma 4.7 *A splice-complete set of pointwise splices has at least two members.*

Proof: In other words we have to show that none of the splice rules can be a primitive for all the splice rules. Assume ρ is a splice rule which works as a primitive for all the splice rules. $\rho(2)$ can not be $\triangle$. If it is, then using an induction similar to the above we can see that position 2 will always have $\triangle$. That is, we will never be able to generate a splice rule σ where $\sigma(2)$ is $\perp$.

Moreover, $\rho(3)$ can not be $\perp$. If it is, then using induction, one can see that all generated splice rules σ will have the property that either $\sigma(3)$ or $\sigma(4)$ is $\perp$. Similarly, $\rho(4)$ can not be $\perp$.

Thus, ρ can have either of $\triangle$, $\triangledown$ or $\perp$ in position 1 and $\perp$, $\triangle$ and $\triangle$ in positions 2, 3 and 4, respectively. But from the previous lemma, we know that an operator specified by such a splice rule can not form a splice-complete set. ■

5 Algebra: Definition

We define the track algebra as the tuple $(\mathcal{T};\ U, \gg, \ll, \leftrightarrow, \rightarrow\leftarrow, \oplus, \asymp, \vdash)$ where

- The domain for the track algebra $\mathcal{T}$, is a set of tracks such that each track $T \in \mathcal{T}$ is a partial map $T : \mathcal{N} \mapsto \mathcal{TAD}$, $\mathcal{N}$ the set of positive integers, and $\mathcal{TAD}$ a set of track atom identifiers. Also, $\mathcal{N} \subset \mathcal{TAD}$. Thus, the index tracks $\mathcal{T}_{\mathcal{N}} \subset \mathcal{T}$ are those tracks $T \in \mathcal{T}$ for which $T : \mathcal{N} \mapsto \mathcal{N}$.

- The primitive operators of the track algebra $\gg, \ll, \leftrightarrow, \rightarrow\leftarrow, \oplus, \asymp$, and $\vdash$ are defined in tables 1, 2, and 3. U is a 0-ary operator such that

$U(1) = 1$ and $U(i)$ is undefined for all $i > 1$.

The above operators are sufficient to derive the remaining operators in tables 2 and 3. Section 4 defines a family of pointwise splices operations and shows that any pointwise splice may be obtained from $\oplus$ and $\asymp$ operators. Since the remaining operators in table 2 ($\uplus$, $\ominus$ and $\odot$) are pointwise splices, they may be derived from $\oplus$ and $\asymp$ operators.

The remaining operators in table 3 ($\dashv$ and $\vdash\dashv$) are defined below. $\dashv$ is defined in terms of the primitive operators, and $\vdash\dashv$ is defined in terms of $\dashv$ and the primitives.

Operation	Signature	Definition
Tail	$\dashv$: $\mathcal{T} \mapsto \mathcal{T}$	$\dashv t \triangleq \vdash (((\vdash t) \gg t) \ll U)$
Extent	$\vdash\dashv$: $\mathcal{T} \mapsto \mathcal{T}$	$\vdash\dashv t \triangleq \dashv ((t \ll (\vdash t)) \gg U)$

Table 7: Derived operations

6 Conclusion

We have presented an algebraic formalism for organizing continuous media data. In the initial sections, we motivated and informally defined the domain and operations in the algebra. Next, we analyzed the family of pointwise splice operations. Finally, we summarized the discussion by formally defining the algebra.

We feel the model can serve many purposes.

- It provides the formal foundations for designing and implementing a track database server.

- It serves as the basis for organizing, synchronizing and creating tem-

poral relationships between time-based elements in hypermedia models [HBvR94, NKN91].

- It serves as a conceptual model for applications such as a console for editing and mixing movies, soundtracks, etc.

References

[All83] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[BGT92] C. Breiteneder, S. Gibbs, and D. Tsichritzis. Modelling of Audio / Video Data. In D. Tsichritzis, editor, *Object Frameworks*, chapter 16, pages 293–310. Centre Universitaire d'Informatique, Universite de Geneve, July 1992.

[EN89] R. Elmasri and S. B. Navathe. *Fundamentals of database systems*. Benjamin/Cummings Publishing Company, Inc., RedWood City, California, 1989.

[GBT92] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Audio / Video Databases: An Object-Oriented Approach. In D. Tsichritzis, editor, *Object Frameworks*, chapter 15, pages 275–292. Centre Universitaire d'Informatique, Universite de Geneve, July 1992.

[HBvR94] L. Hardman, D.C.A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, 37(2):50–62, February 1994.

[LG91] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):30–44, April 1991.

[NKN91] S. R. Newcomb, N. A. Kipp, and V. T. Newcomb. HyTime: The Hypermedia/Time-based Document Structuring Language. *Communications of the ACM*, 34(11):67–83, November 1991.

[Poh92] K. C. Pohlmann. *The Compact Disc Handbook*. The Computer Music and Digital Audio series. A-R Editions, 2nd edition, 1992.

[TL82] D. C. Tsichritzis and F. H. Lochovsky. *Data Models*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[Wal91] G. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4):30–44, April 1991.