Data Structure Properties for Scientific Computing

An Algebraic Topology Library

René Heinzl*
Institute for Microelectronics, TU Wien,
Gußhausstraße 27-29, Vienna, Austria

ABSTRACT

Cell and complex properties are introduced in order to derive a common specification environment for properties of data structures. Only topological properties are used, thereby separating the actual data storage structure from the stored data. Several theoretical topological property concepts are introduced, and traversal and boundary operations are presented and accompanied by selected examples.

Categories and Subject Descriptors

I.1 [**Computing Methodologies**]: SYMBOLIC AND ALGEBRAIC MANIPULATION; D.2.11 [**Software Engineering**]: Architectures—*Data abstraction, domain-specific architectures*

Keywords

Topology, algebraic topology, data structures, generic programming

1. INTRODUCTION

Efficient representation and manipulation of scientific computing's problem domain require data structure concepts supporting arbitrary dimensions and topological spaces.

One prominent example of flexible data structure manipulation is given by the advent of the C++ STL and the separation of access to data structures and algorithms by means of iterators. Iterators have become one of the key elements of modern programming because they enable formulation of algorithms independent of the data structure, which in turn enables exchangeable data structures. Up to now, this approach has focused on sequence, associative, and graph data structures.

The primary objective of the approach presented here is to extend this flexibility to arbitrary data structures and hence enable the treatment of cell complices of arbitrary dimensions. To this end, the extraction and classification of common data structure properties is of utmost importance. Finally the specification of a common protocol for data structure properties is presented. Several libraries which are already available specify common properties of data structures for different areas of application[1] and are cited as examples:

```
gil::image_view <..>        container;
mtl::dense2D <..>           container;
boost::adjacency_list <..> container;
```

A necessary first step is to create a distinct categorization for different types of data structures. The approach presented here uses the dimension of a cell to create a data structure hierarchy. Sequence containers, e.g., STL container, are categorized as 0-dimensional cell storage containers, whereas graph libraries, e.g., Boost Graph Library (BGL [1]), are classified as 1-dimensional cell containers. Higher dimensional cell containers are commonly called grids or meshes.

The other classification property is related to a container's topology, or more generally to a complex's topology. This property enables the distinction between dense and sparse (sequence and associative) structures.

Sections 2–3 therefore introduce basic theoretical classification concepts, while Section 4 presents the library implementation of the given concepts within the GSSE [2–4].

2. TOPOLOGICAL SPACES

The following section introduces concepts of order theory to formalize the combinatorial structure of cells and the global structure of cell complices [5].

DEFINITION 1 (PARTIAL ORDER ON TOPOLOGICAL SPACES). *Let $(X, \mathcal{T})$ be a Hausdorff space. For any $x, y \in X$, a binary relation $\leq$ on X: $x \leq y$, if and only if $x \in cl(y)$ is defined, where $cl(y)$ represents the closure for a set within a topological space, where a closure is the intersection of all closed sets containing y. Then this binary relation is a partial order.*

This definition translates a topological space $(X, \mathcal{T})$ into a partially ordered set $(P, \leq)$. Figure 1 illustrates some types of ordered sets and the following terms which are used to describe them:

- If the order is total, so that no two elements of P are incomparable, then the ordered set is a totally ordered set, called a *chain*. A formal definition is subsequently given in Section 3.
- If no two elements are comparable unless they are equal, then the ordered set is an *anti-chain*.
- If P has two or more incomparable elements, then the ordered set is a *partially ordered set* or *poset*. One example of an incomparable expression is $\{a\}$ with $\{b, c\}$.

*Contact information: heinzl@iue.tuwien.ac.at

POOSC '09, July 7 2009, Genova, Italy

[1] Boost Graphics Image Library (GIL [6]); Matrix Template Library (MTL [7])

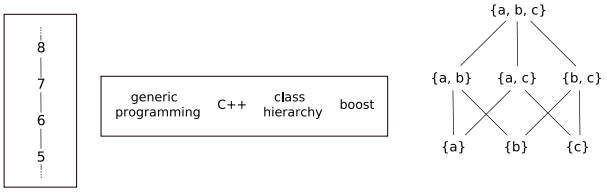

Figure 1: Left: Integers form a chain, totally ordered by $\leq$. Middle: Incomparable items forming an anti-chain. Right: The power-set of $\{a, b, c\}$ ordered by $\leq$ as a partially ordered set.

The concept of *covering* is used in graphical representations of partial orders. For $x, y \in P$ ordered by $\leq$, it is said x is covered by y (written $x \prec y$), if $x < y$ and for any $z \in P$, $x \leq z < y$ implies $x = z$. This means that there is no element of P "between" x and y.

DEFINITION 2 (HASSE DIAGRAM). *A Hasse diagram of an ordered set is a graph in which:*

- *Each node corresponds to an element of the set,*
- *Each edge corresponds to a covering relation between the nodes it connects, and*
- *If x is covered by y ($x \prec y$), then the node for x is drawn in a lower position than the node for y.*

It can be seen that in interpreting diagrams, it does not matter whether one node is above or below another unless there is a monotonic path between them; and that if there is a monotonic path from y through one or more nodes down to x, there is no separate edge directly from y to x, as can be seen in Figure 1.

2.1 Cell Topology

The order concepts which have just been introduced for sets can be used to formalize the internal structure of an arbitrary p-dimensional cell, e.g., using the Hasse diagram. A simplex cell is thereby distinguishable from a cuboid cell type in several dimensions, as depicted in Figure 2 and Figure 3, respectively.

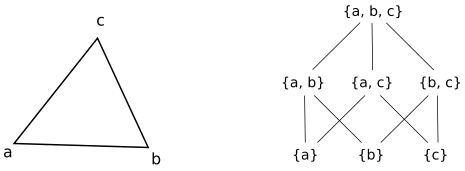

Figure 2: Cell topology of a simplex cell.

Furthermore, sub-cells, such as the corresponding edges and facets and their direct relations to the cell, can be identified. An example of extracting all edges of the simplex cell which corresponds with the middle layer of the Hasse diagram can also be seen in Figure 2. Another noteworthy fact is that the topological structure and the corresponding order hierarchy is invariant with respect to the type of cell used. The $p-1$-layer of a cuboid cell represents the edges of this type of cell, as illustrated in Figure 3.
As a higher dimensional example, a three-dimensional simplex example is provided in Figure 4.

Here, the $p-1$-cells are facets, and, in this particular example, triangles. If the actual dimension of the p-cell is

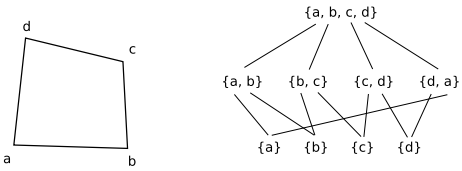

Figure 3: Cell topology of a cuboid cell.

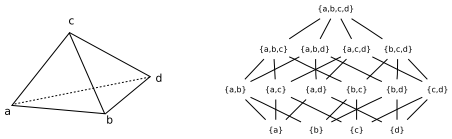

Figure 4: Cell topology of a 3-simplex cell.

used and the $p-n$ cells are derived only by means of the order structure, dimensionally independent algorithms can be used.

2.2 Complex Topology

In contrast to the internal cell topology, a cell complex requires adjacence information of cells, with different possibilities of efficient storage of this information [8, 9]. GSSE includes an abstract means of storing various types of cells orthogonally, based on the cell topology of the complex topology. Figure 5 illustrates the complex topology of a 2-simplex cell complex where the bottom sets are now cells. The rectangle in the figure marks the relevant cell number.

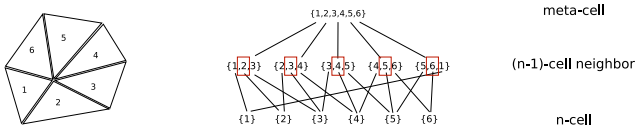

Figure 5: Complex topology of a simplex cell complex.

The topology of the cell complex is only available locally because of the fact that the top set can have an arbitrary number of elements. In the approach presented here this property is called sparse complex topology. The term *meta-cell* is used to describe various subsets with a common name. In other words, there can be an arbitrary number of triangles in this cell complex attached to the innermost vertex.

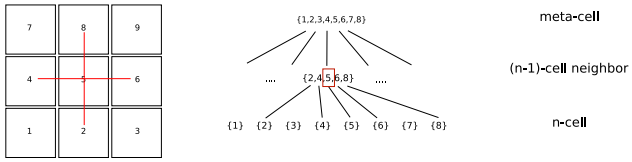

Figure 6: Complex topology of a cuboid cell complex.

Figure 6 presents the complex topology of a cuboid cell complex. The rectangle in the figure marks the main cell (5) under consideration. As can be seen, the number of attached

cells is constant inside the space. The topology of the cell complex can thereby be used as a globally known property, associated with a dense complex topology space.

3. ALGEBRAIC TOPOLOGY

The previous sections introduced combinatorial concepts for cell complex representations and abstract classification mechanisms to manage scientific data. This section focuses more on the details of algebraic properties of linear mappings by a procedure to associate a sequence of Abelian groups or modules with, e.g., a topological space. Briefly, topological elements are called cells, and their dimensionality is expressed by adding the dimension, for example a 3-cell for a volume, a 2-cell for surface elements, a 1-cell for lines, and a 0-cell for vertices. Only a few concepts are introduced in this section [10, 11], as the area of computational topology [12, 13] is a complex and emerging part of scientific computing in its own right.

The motivation for this section is to retain the structure of geometrical objects for computational mechanisms, because the recovery of lost structural information of objects has proven to be a very complex and difficult task.

3.1 Chains

In order for the elements of a specific dimension of a cell complex, e.g., all edges, to be able to be used in a computational manner, a mapping of the p-cells onto an algebraic structure is needed. An algebraic representation of the collection of cells with a given orientation is thus made available. Whereas the cell topology is concerned with the internal structure of a given cell, the chain concept acts on certain p-cells. A formal definition for a p-chain is given by:

DEFINITION 3 (P-CHAIN). *A p-chain c_p defined over a cell complex $\mathfrak{K}$ and a vector space $\mathcal{V}$ is a formal sum*

$$c_p = \sum_{i=1}^{n_p} w_i \tau_p^i \qquad \tau_p^i \in \mathfrak{K}, w_i \in \mathcal{V} \tag{1}$$

such that the operation is closed under orientation reversal:

$$\forall \tau_p^i \in c_p \text{ there is } -\tau_p^i \in c_p \tag{2}$$

Thus, two p-chains can be added, or a p-chain can be multiplied by a scalar. In addition, p-chains support algebraic-topological operations, including the boundary and coboundary operations. Based on these concepts, a cell complex can be seen as a formal structure where cells can be added, subtracted, and multiplied. A structure-relating map between sets of chains C_p is therefore now introduced[2].

DEFINITION 4 (BOUNDARY HOMOMORPHISM). *Let $\mathfrak{K}$ be a cell complex and $\tau_p^i \in \mathfrak{K}, \tau_p^i = \{k_0, k_1, .., k_p\}$. The boundary homomorphism $\partial_p : C_p(\mathfrak{K}) \to C_{p-1}(\mathfrak{K})$ is:*

$$\partial_p \tau_p^i = \sum_i (-1)^i [k_0, k_1, .., \tilde{k}_i, ...k_n] \tag{3}$$

where $\tilde{k}_i$ indicates that k_i is deleted from the sequence.

This can be seen as a boundary operator that maps p-chains onto the $p-1$-chains in their boundary. It should not be confused with the geometric boundary of a point set. This algebraic-topological operation defines a $(p-1)$-chain in terms of a p-chain. It is compatible with the additive and the external multiplicative structure of chains and builds a linear transformation:

$$C_p \xrightarrow{\partial_p} C_{p-1} \tag{4}$$

Therefore, the boundary operator can be used linearly

$$\partial \left(\sum_i w_i \tau_p^i \right) = \sum_i w_i (\partial \tau_p^i) \tag{5}$$

which means that the boundary operator can be used separately for each cell. The cell complex properties can be easily calculated by means of chains. 3-cells intersect in 2-cells or have an empty intersection. This operation can be described by the boundary operator ∂c_p on the cell complex and the corresponding orientation induced on it.

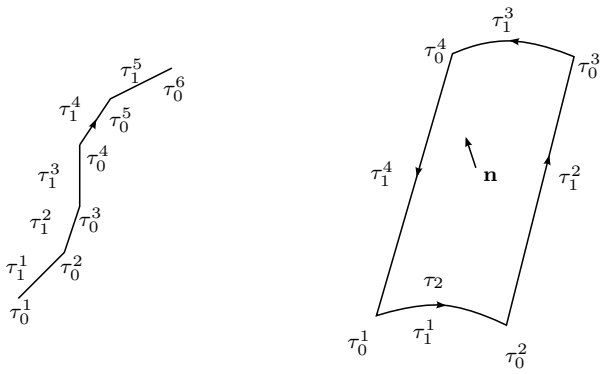

Figure 7: Representation of a 1-chain with boundary (left) and a 2-chain with boundary (right).

Figure 7 depicts two examples of 1-chains, 2-chains, and an example of the boundary operator. Applying the appropriate boundary operator to the 2-chain example:

$$\partial_2 \tau_2 = \tau_1^1 + \tau_1^2 + \tau_1^3 + \tau_1^4 \tag{6}$$

$$\partial_1 \left(\tau_1^1 + \tau_1^2 + \tau_1^3 + \tau_1^4 \right) = \tau_0^1 + \tau_0^2 - \tau_0^2 + \tau_0^3 - \tau_0^3 + \tau_0^4 - \tau_0^4 - \tau_0^1 = 0 \tag{7}$$

3.2 Cochains

Before the introduction of the concept of chains, only the simple structure of a cell complex was available. The cell complex only contains the set of cells and their connectivity. The introduction of the chain concept provides the concept of a collection of cells and the corresponding algebraic structure. Chains can be seen as mappings from oriented cells as part of a cell complex to another space. This definition establishes an algebraic access of computational methods of handling the concept of a cell complex.

In addition to cell complices, scientific computing requires the notation and access mechanisms to global quantities related to macroscopic p-dimensional space-time domains. This collection of possible quantities, which can be measured, can then be called a field. It permits the modeling of these measurements as a field function that can be integrated on arbitrary p-dimensional (sub)domains. An important fact which has to be stated here is that all quantities which can be measured are always attached to a finite region of space. A field function can then be seen as the abstracted process of measurement of this quantity [9, 14]. The concept of cochains allows the association of numbers not only with single cells, as does that of chains do, but also with collections of cells. Briefly, the necessary requirements are that

[2]This map is restricted to simplex cells. A more general mechanism related to the boundary operation is given in Section 2.1

this mapping is not only orientation-dependent, but also linear with respect to the assembly of cells, modeled by chains. A cochain representation is the global association of quantities with subdomains of a cell complex, which can be built arbitrarily to discretize a domain. Physical fields therefore manifest on a linear assembly of cells.

DEFINITION 5 (COCHAINS [15]). *Linear transformations σ of the p-chains into the field $\mathbb{R}$ of real numbers form a vector space $c_p \xrightarrow{\sigma} \mathbb{R}$ and are called a vector valued p-dimensional cochain, or p-cochain.*

The space of all linear mappings on c_p is denoted by C^p, where the elements of C^p are called cochains. Cochains express a representation of fields over a discretized domain $\mathfrak{K}$. Addition and multiplication by a scalar are defined for the field functions and thus also for cochains. To extend the expression possibilities, coboundaries of cochains are introduced.

DEFINITION 6 (COBOUNDARY). *The coboundary δ of a p-cochain is a $(p+1)$-cochain defined as:*

$$\delta c^p = \sum_i v_i \tau_i, \quad \text{where} \quad v_i = \sum_{b \in \text{faces}(\tau_i)} \sigma(b, \tau_i) c_p(b) \quad (8)$$

Thus, the coboundary operator assigns non-zero coefficients only to those $(p+1)$ cells that have c_p as a face. As can be seen, δc_p depends not only on c_p but on how c_p lies in the complex $\mathfrak{K}$. This is a fundamental difference between the two operators ∂ and δ. An example is given in Figure 8 where the coboundary operator is used on a 1-cell. The coboundary of a p-cochain is a $p+1$ cochain which assigns to each $(p+1)$ cell the sum of the values that the p-cochains assigns to the p-cells which form the boundary of the $(p+1)$ cell. Each quantity appears in the sum multiplied by the corresponding incidence number.

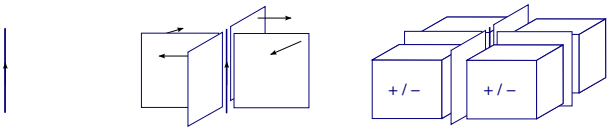

Figure 8: Cochain complex with the corresponding coboundary operator: $\mathfrak{K}^1 \xrightarrow{\delta} \delta\mathfrak{K}^1 \xrightarrow{\delta} \delta\delta\mathfrak{K}^1 = 0$

3.3 Boundary Operators

The concept of chains transforms the properties of a cell complex directly into a computationally manageable algebraic structure. This mechanism can then be used to derive different relations between the cells, such as incidence, adjacence, and boundary operations.

The given boundary operator, introduced in Section 3.1, lacks generality, because the cell topology, see Section 2.1, can be arbitrarily complex, e.g., the given boundary homomorphism already has to be extended if a cube cell is used instead of a simplex cell. Therefore the given poset notation of the cell topology is used to introduce a more general boundary mechanism which can be easily converted into a computationally efficient operation. The boundary operator can then be used to traverse the levels of the poset, e.g., a three-dimensional simplex, illustrated in Figure 9.

As can be seen, the boundary operator simply decreases the layer of evaluation within the cell topology poset. The boundary operator ∂ transforms p-chains into $p-1$-chains, which is compatible with the addition and external multiplication of chains.

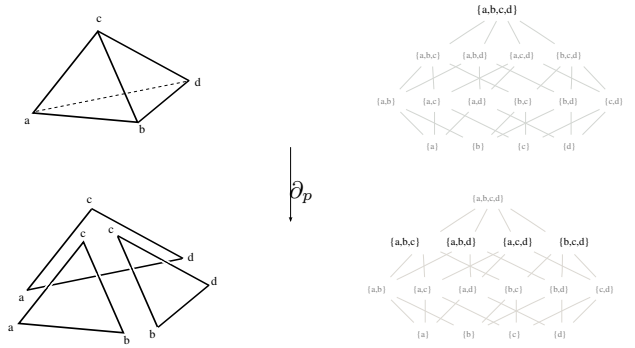

Figure 9: Boundary operator applied onto a 3-simplex poset.

3.4 Application of Chains and Cochains

The concepts of chains and cochains coincide on finite complices [11]. Geometrically, however, C_p and C^p are distinct [15] despite an isomorphism. An element of C_p is a formal sum of p-cells, where an element of C^p is a linear function that maps elements of C_p into a field. Chains are dimensionless multiplicities, whereas those associated with cochains are physical quantities [9]. The extension of cochains from single cell weights to quantities associated with collections of cells is not trivial and makes cochains very different from chains, even on finite cell complices. Nevertheless, there is an important duality between p-chains and p-cochains.

For a chain $c_p \in C_p(\mathfrak{K}, \mathbb{R})$ and a cochain $c^p \in C^p(\mathfrak{K}, \mathbb{R})$, the integral of c^p over c_p is denoted by $\int_{c_p} c^p$, and integration can be regarded as a mapping, where n represents the corresponding dimension:

$$\int : C_p(\mathfrak{K}) \times C^p(\mathfrak{K}) \to \mathbb{R}, \qquad \text{for } 0 \le p \le n \quad (9)$$

Integration in the context of cochains is a linear operation: given $a_1, a_2 \in \mathbb{R}, c^{p,1} c^{p,2} \in C^p(\mathfrak{K})$ and $c_p \in C_p(\mathfrak{K})$, reads

$$\int_{c_p} a_1 c^{p,1} + a_2 c^{p,2} = a_1 \int_{c_p} c^{p,1} + a_2 \int_{c_p} c^{p,2} \quad (10)$$

Reversing the orientation of a chain means that integrals over that chain acquire the opposite sign

$$\int_{-c_p} c^p = -\int_{c_p} c^p, \quad (11)$$

using the set of p-chains with vector space properties $C_p(\mathfrak{K}, \mathbb{R})$, e.g., linear combinations of p-chains with coefficients in the field $\mathbb{R}$.

4. AN ALGEBRAIC TOPOLOGY LIBRARY

By identifying data structures with finite topological spaces, or more specifically with cell complices, concepts from algebraic topology, such as complex type, cell type, and the dimension, are available for common properties to be studied.

The first determination property is related to local information, the dimension of a cell, and enables the following categorization:

- 0-cell: sequence data structure, e.g., STL container
- 1-cell: graph library, images library, e.g., BGL, GIL
- higher dimensional-cell: grid and mesh data structure

The second classification property is related to global information, the complex property which classifies the cell storage mechanisms:

- dense storage: sequence data structures or Cartesian grids
- sparse storage: associative containers and meshes

A flexible specification of data structures requires a variable number of properties to be specified, hence a meta data structure specification protocol is used. The Boost Phoenix [16] environment concept already specifies a general protocol to handle and manipulate arbitrary arguments for functional programming:

```
mpl::map<
 mpl::pair<gsse::env_dim,     mpl::int_<2> >
,mpl::pair<gsse::env_cell,    gsse::tag_simplex>
,mpl::pair<gsse::env_complex,gsse::tag_sparse>
,mpl::pair<gsse::env_cont,    gsse::tag_vector>
> PropertyEnv;
```

The final data structure is then generated by applying a simple meta-function generator

```
gsse::create<PropertyEnv>::type container;
```

which uses a direct mapping of all data structure properties to either select an existing data structure, e.g., STL containers, or to create appropriate data structures, e.g., for higher dimensional cell complices.

4.1 Data Structure Specification

The STL containers `std::vector<>`, `std::list<>`, are described topologically as 0-cell complex, a POD `double` type as cell property, a dense complex topology, and finally the underlying storage container.

```
mpl::map<
 mpl::pair<gsse::env_dim,     mpl::int_<0> >
,mpl::pair<gsse::env_cell,    double>
,mpl::pair<gsse::env_complex,gsse::tag_dense>
,mpl::pair<gsse::env_cont,    gsse::tag_vector>
> PropertyEnv;
```

Due to compile-time evaluation no run-time overhead for the generated data structures is introduced. By using the topological space classification of cell complex properties, STL's associative containers `std::map<>` and `std::unordered_map` are classified by a sparse complex topology:

```
mpl::map<
 mpl::pair<gsse::env_dim,     mpl::int_<0> >
,mpl::pair<gsse::env_cell,    double>
,mpl::pair<gsse::env_complex,gsse::tag_sparse>
,mpl::pair<gsse::env_cont,    gsse::tag_map>
 > PropertyEnv;
```

For higher dimensional cell complices, cell and complex cell properties have to be specified, e.g., a sparse 3-simplex cell complex (tetrahedral mesh):

```
mpl::map<
 mpl::pair<gsse::env_dim,     mpl::int_<3> >
,mpl::pair<gsse::env_cell,    gsse::tag_simplex>
,mpl::pair<gsse::env_cont,    gsse::tag_vector>
,mpl::pair<gsse::env_complex,gsse::tag_sparse>
> PropertyEnv;
```

To switch the application from a sparse tetrahedral mesh to a grid (dense cuboid grid), the cell and complex properties have to be changed:

```
mpl::map<
 mpl::pair<gsse::env_dim,     mpl::int_<3> >
,mpl::pair<gsse::env_cell,    gsse::tag_cuboid>
,mpl::pair<gsse::env_cont,    gsse::tag_vector>
,mpl::pair<gsse::env_complex,gsse::tag_dense>
> PropertyEnv;
```

In this fashion cell complices of arbitrary dimension can thereby be created. The current implementation is restricted to enabling 0D up to 8D due to long compile times and the large memory footprint during compilation for higher dimensions.

4.2 Traversal

Traversal of elements is realized by a Boost Phoenix traversal actor `gsse::traverse<>()[]`. As an example, the vertex traversal (topological accessor: `AT_vx`) for an arbitrary dimensional cell complex is presented in the next code snippet:

```
gsse::traverse<AT_vx>()
[
  gsse::traverse() [ _1 ]
](  cell_complex );
```

Another example is given by traversing all edges of a cell complex which cell dimension is greater then 0D. Here the topological accessor for edges `AT_ee` is used:

```
gsse::traverse<AT_ee>()
[
  gsse::traverse() [ _1 ]
](  cell_complex );
```

For arbitrary dimensional programming, two possible accessor mechanisms are available: the already presented topological object accessors (vertex: `AT_vx`, edge: `AT_ee`, facet: `AT_ft`, cell `AT_cl`) or direct dimensional accessors (vertex: 0D, edge: 1D). But for facets and cells, the dimensional correlation is not possible for arbitrary dimensional cell complices. To solve this issues, GSSE always uses name tags, e.g. for arbitrary dimensional cell traversal:

```
gsse::traverse<AT_cl>()
[
  gsse::traverse() [ _1 ]
](  cell_complex );
```

If an algorithm requires the actual dimension or an inter-dimensional topological object accessor has to be calculated, meta-conversion functions are available:

```
gsse::meta::key_2_dim<cc, gsse::AT_ee>::value;
gsse::meta::dim_2_key<cc, DIM>::type;
```

The cochain concept is implemented by means of an additional Phoenix actor, given in the following code snippet by `acc`. Due to the topological structure of the cochains, cochains can also be traversed.

```
gsse::traverse<AT_cl>()
[
  gsse::traverse() [ acc += 12 ]
](  cell_complex );
```

Compared to existing libraries, e.g., BGL or GrAL [17], GSSE's data structure specification and traversal operations do not impose any runtime overhead, hence in all benchmarks no runtime difference was measured, thus GSSE can compete again with other libraries [18].

4.3 Boundary Operations

As introduced in Section 3.3 algebraic topology concepts enable additional operators on data structures, such as boundary and co-boundary operators, e.g., vertex-on-cell and cell-on-vertex operators. These operators benefit from the abstract topological specification and operate on spaces of arbitrary dimension and topology. Given an N-dimensional cell, the 2-boundary is calculated by:

```
gsse::Boundary<N, 2, gsse::tag_cell_simplex>
   boundary_nD_2_t boundary_nD_2;

cont_result = boundary_nD_2(container_cells[1]);
```

In case a 4D hyper-cube cell complex is generated, this operation results in a quadrilateral-on-cell operation. Convenience mechanisms are available, which use direct dimensional accessors for boundary cells and source-cells:

```
typedef result_of::vx_on_cell<Complex>::type
   vx_on_cl_type;
typedef result_of::vx_on_cell<Complex>::result
   vx_on_cl_result;
vx_on_cl_type vx_on_cl;

vx_on_cl_result result =
   vx_on_cl ( container_cells[1] );
```

Co-boundary operators act globally on a complex and thus require additional information. The following example presents an edge-on-vertex operation, where the operator source is stated by a 0-cell (vertex), whereas the co-boundary dimension is stated by the relative cell dimension, in this case `+1` (edge). A corresponding meta-function calculates the target container for this operation:

```
typedef coboundary<0, 1 ,gsse::tag_cell_simplex>
   cobnd_sD_1_t;

cobnd_sD_1_t cobnd_sD_1;
result = cobnd_sD_1(cell,
                    container_cell_on_vertex,
                    container_vertex_on_cell);
```

The boundary operator results are also traversable and can be used in any algorithm.

5. CONCLUSION

By using order and topological space order concepts, data structure properties are identified with cell and complex topologies. Algebraic topology concepts introduce chain and cochain bodies and enable efficient boundary and coboundary operations. Data structures are thereby separated into the structural part and the data storage part. The library implementation then offers generic traversal and abstract boundary operators. A common specification for data structure properties of arbitrary dimension and topological cell types enables the efficient development of generic algorithms and applications in the field of scientific computing. Using this library, several other libraries have been developed, e.g., a polynomial library, an unstructured mesh generator, and an hp-finite-element application for arbitrary dimensions and topologies.

6. ACKNOWLEDGMENTS

I want to thank Philipp Schwaha and Franz Stimpfl for their continuous support and Prof. Siegfried Selberherr for the resources at the Insitute of Microelectronics. This work has been supported by the Austrian Science Fund FWF, project P19532-N13.

7. REFERENCES

[1] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

[2] R. Heinzl. *Concepts for Scientific Computing*. Dissertation, Technische Universität Wien, Austria, 2007.

[3] R. Heinzl, P. Schwaha, and S. Selberherr. A High Performance Generic Scientific Simulation Environment. In B. Kaagström et al., editor, *Lecture Notes in Computer Science*, volume 4699/2007, pages 996–1005. Springer, Berlin, June 2007.

[4] R. Heinzl, P. Schwaha, F. Stimpfl, and S. Selberherr. Parallel Library-Centric Application Design by a Generic Scientific Simulation Environment. In *Proc. of the POOSC*, Paphos, Cyprus, July 2008.

[5] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge, 1990.

[6] Boost. *Boost Graphics Image Library (GIL)*, 2005. http://www.boost.org/.

[7] P. Gottschling and D. Lindbo. Generic Compressed Sparse Matrix Insertion: Algorithms and Implementations in MTL4 and FEniCS. In *POOSC 2009 Workshop at ECOOP09*, ACM Digital Library, 2009.

[8] P. Gross and P. R. Kotiuga. *Electromagnetic Theory and Computation: A Topological Approach*. Cambridge University Press, 2004.

[9] C. Mattiussi. The Geometry of Time-Stepping. In F. L. Teixeira, editor, *Geometric Methods in Computational Electromagnetics, PIER 32*, pages 123–149. EMW Publishing, Cambridge, Mass., 2001.

[10] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[11] J. Hocking and G. Young. *Topology*. Addison-Wesley, Dover Publications, New York, 1961.

[12] A. J. Zomorodian. Topology for Computing. In *Cambridge Monographs on Applied and Computational Mathematics*, 2005.

[13] T. Dey, H. Edelsbrunner, and S. Guha. Computational Topology. In J. E. G. B. Chazelle and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, Contemporary Mathematics. Providence, RI, USA, 1998.

[14] E. Tonti. The Reason for Analogies between Physical Theories. *Appl. Math. Modelling*, 1(1):37–50, 1976/77.

[15] P. Bochev and M. Hyman. Principles of Compatible Discretizations. In *Proc. of IMA Hot Topics Workshop on Compatible Discretizations*, volume IMA 142, pages 89–120. Springer, 2006.

[16] Boost. *Boost Phoenix 2*, 2006. http://spirit.sourceforge.net/.

[17] G. Berti. *Generic Software Components for Scientific Computing*. Dissertation, Technische Universität Cottbus, 2000.

[18] R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser. Performance Aspects of a DSEL for Scientific Computing with C++. In *Proc. of the POOSC Conf.*, pages 37–41, Nantes, France, July 2006.