https://doi.org/10.1142/9789811239922_0012

Chapter 12

A Qualitative Reasoning Model for Software Testing, based on Combinatorial Geometry

Spyros Xanthakis

Athens University of Applied Science, Athens, Greece

Emeric Gioan

LIRMM, University of Montpellier, CNRS, Montpellier, France

12.1 Introduction

Let us first give a short summary of the paper. We introduce an operational *qualitative model* for numerical algorithms. This model enables *qualitative algorithmic reasoning* during the unit testing process, supported by an automatic generation of boundary test data. It contains the spatial encoding of all the functionally equivalent regions (called *homodromies*). The model, viewed as a qualitative abstraction of the algorithm, is automatically generated thanks to multiple targeted executions of its instrumented basic conditions, that permit the interpolation of a set of linear equations and their corresponding *hyperplanes*. All feasible paths (when all basic conditions are linear) are identified in the form a *Ternary Decision Tree*. Each feasible execution path corresponds to a geometric region supported and/or delimited by a finite set of hyperplanes of the vector space. *Oriented Matroids* allow the encoding of the relative positions of all such regions of various dimensions, in a purely combinatorial way, using a finite signed set system, supported by a rich mathematical theory. This model uses a *qualitative space algebra* and enables *qualitative reasoning*: regions are identified according to qualitatively valued inputs, which are, in their turn, propagated through them. The proposed model permits a global/local envisionment

of the algorithmic behaviour in an abstract level. The cocircuits of the underlying oriented matroid play the role of testing building blocks: the combination of their numerical coordinates enables the automatic generation of limit boundary data that lies at the boundary of any critical surface of any dimension.

Let us now present the paper and its context in more details. From the very beginning, the software engineering community tried to adopt artificial intelligence (AI) techniques for assisting an especially time-consuming and complex phase of the software development process: unit testing. *Unit testing* is destinated to assess the units programmed during the implementation phase (a C-procedure, a Python method). During this phase, the tester, who is often the developper, has to design test data (according to unit testing criteria), prepare the right execution environment, execute the (possibly instrumented) software, filter the outputs, analyse and validate the final results. Testing criteria could be functional, like, "Test what happens when alarm is on", "Check the answer of the system when the temperature is very high", or structural like, "Check what happens if a loop statement is not executed", or "Be sure that, at the end of unit testing, all statements have been executed at least once", etc.

Evolutionary (or *search based*) techniques, initially devoted to the AI area, have nowadays proved their efficiency in software testing, and particularly in the automatic test data generation. A first application of evolutionary techniques for unit testing was proposed by one of the authors of this paper [1, 2]. It consisted in expressing a structural test data objective as an optimization problem handled by a conventional steady state genetic algorithm (GA) coupled with an elementary data flow analysis. Many works followed that improved the performances and extended the range of techniques and applications; they are now grouped under the name *evolutionary testing* [3–8] which is no more limited to the use of GAs but encompasses a wide spectrum of AI techniques. More broadly, evolutionary (or search based) software engineering, can now tackle with different testing tasks: functional testing, integration testing, mutation testing, regression testing and test case planning [5, 9–13]. Test data generation, and especially limit (or boundary) test data, constitute a major open issue in the software development process. Our testing data generation philosophy does not rely on a specific structural coverage objective (i.e. cover all the statements), nor uses an a priori testing model (i.e. a state machine or a control graph), nor a random/heuristic/evolutionary technique. Under certain assumptions

that will be presented in the beginning of the next section (briefly, for a constant-time algorithm based on linear inequalities), it handles, formally and exhaustively, all the feasible execution paths of the source code.

In our view, existing automatic testing generation techniques, do not dispose of a sufficient high level of abstraction to assist testers intelligently. The testing process cannot be reduced to an input/output comparison task. It often involves designing activities (design of the testing strategy and its deployement, redesign a defectuous code, etc.) but also reverse engineering tasks (assisted by many visualisation and/or debugging tools, etc.). All those activities necessitate a minimum of understanding and reasoning about the actual and/or the required behaviour of an algorithm. We will try to illustrate our point by a (non exhaustive) list of some questions that may arise during testing (assuming that inputs are endowed with an absolute order relation). Questions of this kind are answered in Sec. 12.4.

- How and in what direction a specific output is impacted by a small change of inputs? Which are the inputs that are influencing a condition in the source code? In what direction could one change the inputs in order to switch its truth value from `False` to `True`?
- What happens at a close neighborhood of a critical point? If the temperature is very high and the pressure is normal will the alarm be triggered? Will it stay triggered even if the temperature raises "indefinitively"? Is it possible to find contiguous regions with different behaviour, and, if yes, what are the limit test data that separate them?
- If one decreases the element of an array or if this element is very small, will it eventually appear at the beginning of an ascending order sorted array? In the Knap Sack problem if one decreases the weight of an already selected item, will the total value increase or decrease? In the Traveler Salesman Problem, if the distance between two cities is decreased, will their orders be closer in the solution?

We argue that all those kind of questions are pervasive in software testing and contain indisputably qualitative reasoning aspects, extensively studied by the qualitative reasoning community in AI, such as *comparative analysis and envisionment*, *order-of-magnitude* and *spatial reasoning* [14–20]. We will illustrate that spatial properties are implicit when reasoning about numerical algorithms. We do not claim that qualitative reasoning exhausts all the mental activities of unit testing (and even less,

of algorithmic reasoning in general), but that it constitutes an essential component.

In our approach, an algorithm is viewed as a *fragmentation* process of the input space considered as a real vector space. It uses sequences of nested and iterated logical conditions in order to fragment the input space into small connected pieces, the fragments. At each point (which corresponds to a specific input) of a fragment is associated a uniform sequence of computations. One of the main tasks of software testing could be compared to the task of checking whether those fragments are well "glued" together. Our central working assumption can thus be formulated as follows: if one is capable, first, of qualitatively expressing the underlying spatial information of the fragments (i.e., contiguity, direction, intersection, inclusion, etc.) and, secondly, qualitatively expressing input values (i.e., small, big, close to, etc.), while ensuring their propagation between and inside those regions, then an algorithmic qualitative reasoning is possible.

We present a qualitative reasoning algorithmic model. Its construction is illustrated in Fig. 12.1. The spatial relations between fragments are expressed using a very active area of combinatorial geometry: Oriented Matroid theory (whose fundamentals are recalled in Sec. 12.3). Qualitative propagation is ensured by an order-of-magnitude algebra. To any cocircuit of the oriented matroid (combinatorial encoding of a 1-dimensional region) is naturally associated the quantitative information (real coordinates) of its location in the space. This is used to produce, automatically, by means of algebraic combinatorial properties, limit test data lying at the boundary of any critical surface of any dimension. Our model necessitates only a simple instrumentation of the basic conditions of the source code. It can be then used for functional/non-functional as well for objective/non objective languages ranging from assembly language to Fortran, C, LISP, C++, Java, Python, etc. It encompasses all applications using mainly numeric inputs, such as avionics, statistics, banking, simulation, robotics, etc. Further comments are given in the conclusion.

12.2 What the model contains

Our model is based on an instrumented source code. In the whole paper, our assumption is that the source code implements a constant-time algorithm, and that it is based on real (decimal) linear (in)equalities, so that all possible conditions executed for all possible inputs amount to evaluate some real linear functions of the input vector space, yielding a finite set of

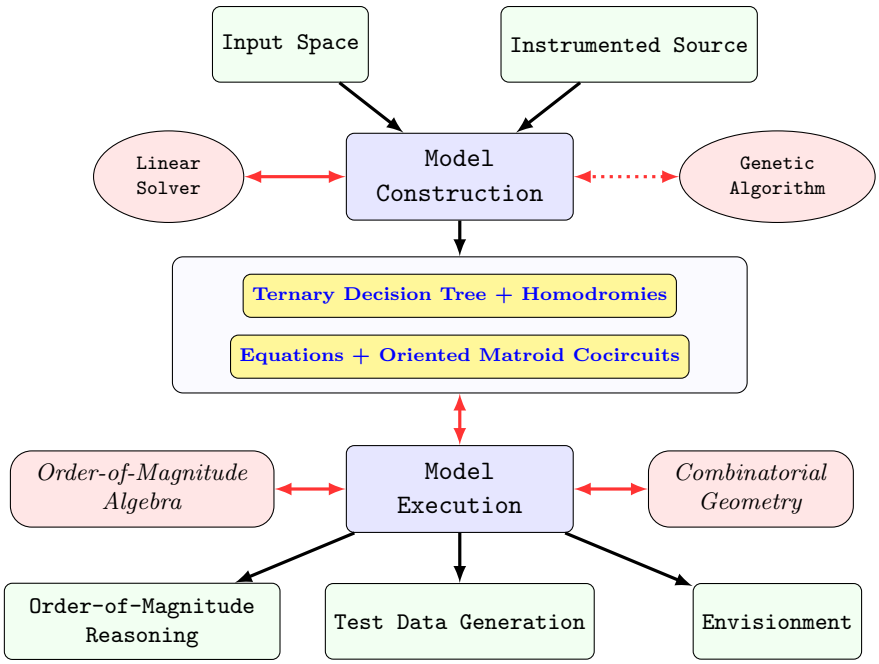


Fig. 12.1 The two main steps of the algorithmic qualitative modelling process: `Model Construction` and `Model Execution`. The *Instrumented Source* code, and its *Input Space* are given. In case of non-linearity a *Genetic Algorithm* can be used to detect a critical point. A rational *Linear Solver* builds a *Ternary Decision Tree* and interpolates the equations (generated by the basic conditions of the source code), their corresponding oriented matroid cocircuits and all the homodromies (geometric regions associated to feasible paths). `Model Execution` will then use *Order-of-Magnitude Algebra* combined with a *Combinatorial Geometry* algebra (playing the role of a qualitative space algebra) enabling the abstract *Execution* and *Envisionment* of the source code behaviour. Finally, quantitative information associated with the cocircuits permits the automatic generation of limit test data that lies exactly at the boundaries (of any dimension) of critical surfaces.

useful real linear functions, as detailed below. In this case, the model is completely well-defined and extensive. In the non-linear case (and in the non-constant-time case), that will not be addressed in this paper, the same technique can be used locally, at a close neighbourhood of a critical point of the input space, by considering appropriate linear inequalities, possibly with an approximation (this can be based on the Jacobian matrix, and/or found by Genetic Algorithms techniques).

We first introduce the content of our model (Sec.12.2.1). Next, we illustrate it on a simple example that will serve as running example throughout this paper. Section 12.2.3 details the technical construction of the model.

12.2.1 *Conditions, Equations, Paths, Super-covectors, and Homodromies*

Let us start with a source code using numeric linear (in)equalities as conditions. When the source code is executed for a given input, a sequence of conditions is executed. We say in that case that these conditions are *sensitized*. To each condition corresponds a truth value, `True` or `False`, observed during the execution path, with respect to the current variable values. We will not focus on the sequence of truth values but on a sequence of signs $+$, $-$, or 0, which we call the *sign-path*, defined as follows. When the condition is executed in the source code, written in the form [`left-hand-side (LHS) comparison-operator right-hand-side (RHS)`], then we record the sign of the value of [`LHS` $-$ `RHS`] (with respect to the current variable values). Intuitively speaking, when the condition is an inequality, the 0 case can be thought of as a "limit case between `True` and `False`". Our model takes into account these limit cases and handles them in a precise geometric combinatorial structure.

According to our assumption, each condition in the sequence consists to checking whether an associated real linear function of the input space has a positive, negative or null value. In what follows, we abusively call this linear function an *equation* (though it is the underlying inequality that is used). Generally, equations do not appear explicitly in the source code (in contrast with our toy example), and the same source condition may yield several equations (in the case of a loop, typically). The set of all resulting equations for all possible executions forms a set E, which is finite (because of the constant-time assumption). It is computed by a tree search based on the sign-path associated to an interpolation technique using a linear solver (see Sec. 12.2.3). We call *equation-path* the sequence of sensitized equations (among E), and with each equation is associated a sign in $\{$`+`, `-`, `0`$\}$ according to the value of its corresponding linear function. A supplementary equation "at infinity" is added when the initial inequalities are affine, in order to use a standard vector space setting; its sign will always be `+` for executions; it will also permit to model the algorithm's behavior "at infinity". We stress the fact that two inputs admit the same sign-path if and only if they admit the same equation-path (though the two sequences of signs can be different). This property ensures that, for each execution path, a unique equation (with a unique sign) will be associated with every basic condition, thus permitting its interpolation.

Finally, each possible execution is encoded by associating each element

of E with either its sign from the equation-path or the *undetermined sign* $\times$ when the equation is not sensitized in the equation-path, yielding what we call a *super-covector* over E, or *scovector* for brevity. Those combinatorial objects can actually be understood as parts of the oriented matroid structure underlying the set E of real equations in the real input vector space, as detailed in Sec. 12.3. Note that two inputs admitting the same scovector do not generally yield the same output (as in our toy example); however they sensitize the same execution path. The converse is not true: two inputs sensitizing the same execution path may admit distinct scovectors (because of the `0` sign of conditions, as noted above). The resulting scovectors yield equivalence classes aggregating inputs that sensitize the same execution path.

The geometric interpretation of the previous equivalence relation is captured by the concept of *homodromy*. Homodromies — from the greek, $\acute{o}\mu o\iota o\varsigma(same)$, $\delta\rho\acute{o}\mu o\varsigma(path)$ — are connected regions of the (geometrized) input space, that exhibit the "same" algorithmic behaviour. They group inputs that are operationally treated in the "same manner". Clearly, homodromies correspond to feasible execution paths and can be compared to a sort of "software phases". Two inputs belong to the same homodromy (we say in that case that they are *homodromic*) if one can connect them with a "continuous" geometric path without changing the sign-path (or, equivalently, the equation-path). As explained above, homodromies are combinatorially encoded by scovectors, and we shall say that the homodromy is the *region* of a scovector encoding the homodromy as above. Let us emphasize this notion as it constitutes the central notion of this work. (Let us mention that the geometric concept of homodromy as defined below could be extended to any source code based on a geometric input space.)

Definition 12.1. We call *homodromies* the path-connected components (of the input space) induced by the equivalence relation based on the sign-path (or, equivalently, the equation-path). Homodromies form a partition of the whole input space called the *fragmentation* of the input space with respect to the source code (see Definition 12.6 below for a combinatorial counterpart).

12.2.2 *Running example*

Suppose that one wishes to perform structural dynamic testing of the following source code, which implements the specifications of an approximative and simplified model of the water transition phase diagram from classical

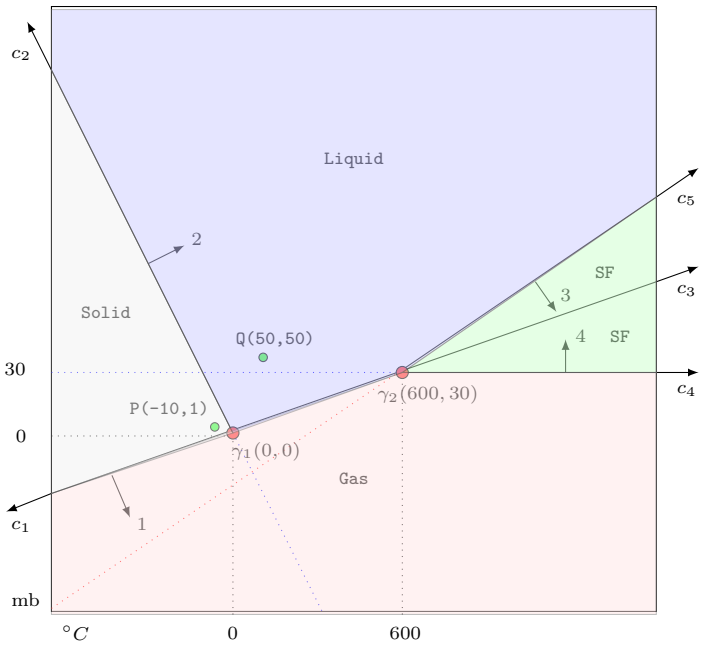


Fig. 12.2 Phase Transition Diagram of the toy algorithm. Four equations, induced by the conditions in the source code, delimit four zones corresponding to distinct outputs (phases). These equations $e_1, \ldots, e_4$, are depicted as 1, 2, 3, 4, and "oriented" by arrows towards their positive side (the points where the equation is positive). The boundary of the `Solid` zone (formed by regions/homodromies yielding the same `Solid` output) is formed by the equations e_1 and e_2 sensitized by the internal point P(`-10,1`). Similarly, the point, Q(`50,50`) (`Liquid` zone) sensitizes its border equations e_1, e_2, e_3. The points γ_1 and γ_2 are intersections of lines corresponding to the equations, and c_1, c_2, c_3, c_4, c_5 denote the useful directions of those lines at infinity.

physics, as illustrated in Fig. 12.2. The `WaterPhases` algorithm takes two decimal inputs: t (between -100.0 and $+1000.0$) and p (between -100.0 and $+100.0$), representing the temperature (in degree Celsius) and the pressure (in bars) of the water, respectively. Then, it outputs its state: `Solid, Liquid, Gas, SuperFluid`.

```
if t >= 20*p:
      if p > 30: return 'SuperFluid'
      else: return 'Gas'
elif 10*t + p <= 0: return 'Solid'
elif t - 10*p < 300: return 'Liquid'
else: return 'SuperFluid'
```

The previous source code uses four basic conditions: `(t >= 20*p)`, `(10*t + p <= 0)`, `(t - 10*p < 300)` and `(p > 30)`. Basic conditions (simply called *conditions*) are expressions separated by a comparison operator {`<=`, `>=`, `==`, *etc.*}. They are uniquely identified by their textual position inside the source code and may appear in many different forms, like, inside compound conditions using logic connectors, and/or at the right hand of boolean assignment statements. Note that, the (basic) conditional statement "`if (t >= 20*p)`" is equivalent to "`if (t - 20*p >= 0)`".

Let's now execute the previous code for the input $t =$ `-10.0` and $p =$ `1.0`, corresponding to a cold temperature of $-10°C$ and a normal atmospheric pressure of 1 *bar*. First condition, `t >= 20*p`, is executed (*sensitized*) yielding the truth value `False` with the negative value $t - 20 * p = (-10) - 20(1) = -30$. The *execution path* follows the `else` alternative and sensitizes the condition `p + 10*t <= 0` which is `True` (since `1+10.(-10)=-99<=0`) and the observed value is negative; the final output will be the `Solid` (ice) phase. In conclusion, the two conditions have been sensitized, yielding the sign-path `[--]`. These conditions can be interpolated (as it will be explained in Sec. 12.2.3) yielding respectively the linear equations $e_1(t, p) = t - 20p$ and $e_2(t, p) = 10t + p$. The *equation path* associated to the point $P(-10, 1)$ will be thus noted 1^-2^-. The resulting *scovector* is [`--xx+`] (the last `+` sign comes from an equation added "at infinity" as mentioned above).

Table 12.1 includes these results for all executions/homodromies of the toy example.

12.2.3 *How the model is built technically*

In this section, we present how the successive executions of the source conditions permit the interpolation of their corresponding (unique) equation inside each homodromy. This is accomplished by a progressive expansion of a Ternary Decision Tree.

Ternary Decision Trees. Let $\mathbf{E} = \{e_i\}_{i=1..n}$ be a set of linear equations on $\mathbb{R}^r$. A *Ternary Decision Tree (TDT)* is a tree where any non terminal node belongs to $\mathbf{E}$ and admits 1, 2 or 3 successors (according to their feasibility) connected with *sign labels* in {`+`, `-`, `0`} (each sign appearing only once). For convenience, one can collapse identical brother subtrees in a unique subtree decorated by the union of the original signs (see Fig. 12.3).

Table 12.1 The 11 homodromies of the toy algorithm (grouped according to their topological dimension), denoted $\Omega_1, \ldots, \Omega_{11}$, their sign-paths and equation-paths, their resulting encoding as scovectors, the corresponding output, and an illustrative boundary test data proposed by the tool (see Sec. 12.4.3). The column "Test data" gives an input yielding the corresponding sign-path. For instance, in the last row, the test data `[600,30]` means that putting the temperature at 600 degrees and the pressure at 30 bars, the algorithm output (the water phase) will be `Gas` and the sensitized homodromy corresponds to the scovector [`0xx0+`].

Ω_i	*Sign-path*	*Equation-path*	*SCovector*	`Output`	`Test data`
1	[--]	1^-2^-	[--xx+]	`Solid`	`[-100,999]`
2	[-+-]	$1^-2^+3^-$	[-+-x+]	`Liquid`	`[599.99,30]`
3	[-++]	$1^-2^+3^+$	[-++x+]	`SuperFluid`	`[600.3,30.02]`
4	[++]	1^+4^+	[+xx++]	`SuperFluid`	`[600.21,30.01]`
5	[+-]	1^+4^-	[+xx-+]	`Gas`	`[-100,-100]`
6	[-0]	1^-2^0	[-0xx+]	`Solid`	`[-100,1000]`
7	[-+0]	$1^-2^+3^0$	[-+0x+]	`SuperFluid`	`[600.1,30.01]`
8	[0+]	1^04^+	[0xx++]	`SuperFluid`	`[600.2,30.01]`
9	[+0]	1^+4^0	[+xx0+]	`Gas`	`[600.01,30]`
10	[0-]	1^04^-	[0xx-+]	`Gas`	`[-100,-5]`
11	[00]	1^04^0	[0xx0+]	`Gas`	`[600,30]`

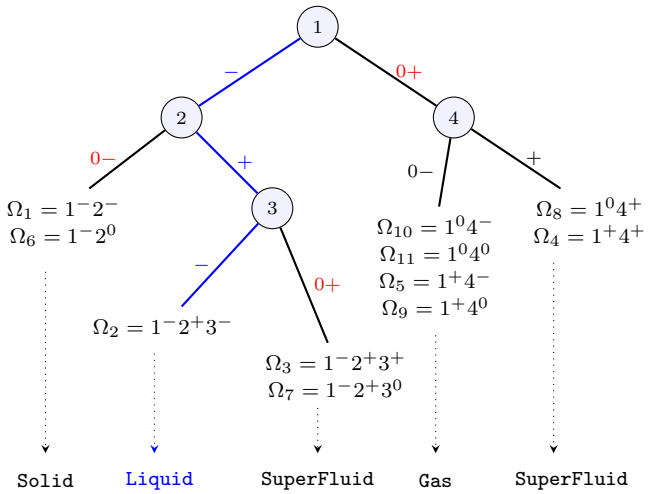


Fig. 12.3 The Ternary Decision Tree (TDT) modelling the algorithmic behaviour of the `WaterPhases` algorithm. Each of the 11 leaves corresponds to a unique homodromy of Table 12.1. The equation-path $\varrho = 1^-2^+3^-$ sensitized by Q(`50,50`) corresponding to the `Liquid` homodromy Ω_2. Note that those equations $1, 2, 3$ form its geometric boundary (see Fig. 12.2). As addressed in Sec. 12.2.3, this equation-path ϱ admits 6 tree alternatives: $\{1^0, 1^+, 1^-2^0, 1^-2^-, 1^-2^+3^0, 1^-2^+3^+\}$. A rational linear solver will produce solutions (or not) for those alternatives, and thus new points to be executed that will eventually sensitize the 4th equation.

Each *leaf* (node with no successor) expresses a geometric region, i.e. a single homodromy, thus a feasible execution path, labeled by its equation-path. A tree using n equations will admit at most 3^n feasible leaves. Their feasibility depends on the intersection pattern of the equations. (Let us mention that this feasability is encoded in the underlying oriented matroid.)

Tree Expansion Algorithm. The tree expansion algorithm follows a conventional tree breadth-first strategy (BFS). At a first stage, a random sampling provides a bootstrap point. Consider for instance that the point $Q = (50,50)$, of the `Liquid` homodromy Ω_2, is the bootstrap point. Q has a sign-path [`-+-`] and sensitizes equations $\{e_1, e_2, e_3\}$. Using the rational linear solver, in a small enough neighborhood of Q, it is now possible to generate a sufficient number of homodromic points enabling their interpolation. In fact, since all points belong to the same homodromy, we know that the condition `(t >= 20*p)` expresses a unique equation of the form: $e_1(t,p) : k_1 t + k_2 p + k_3 = 0$ (since we use two inputs). In our example, the execution of two additional homodromic points, yields two more values for the equation e_1, and allows the evaluation of the coefficients k_1, k_2, k_3, and thus the exact mathematical expression of $e_1(t,p) = t - 20p$. The same interpolation is done by observing the values of the conditions `(10*t + p <= 0)` and `(t - 10*p < 300)`, yielding the equations $e_2(t,p) = 10t + p$ and $e_3(t,p) = t - 10p - 300$. The equations having been identified, it is now possible to reconstitute the first equation-path path of the TDT: $\varrho = 1^-2^+3^-$ (see the path in Fig. 12.3), yielding the scovector [`-+-x+`]. At a second stage, follows the generation of the 6 *tree alternatives* of ϱ: $\{1^0, 1^+, 1^-2^0, 1^-2^-, 1^-2^+3^0, 1^-2^+3^+\}$ (see Fig. 12.3). Each tree alternative corresponds to a subpath of ϱ starting from the root, where the last node has been extended with only one branch different from the original one. Each alternative corresponds to a system of (strict) linear (in)equalities submitted to a rational linear solver. Starts now, a classic iterative process of a ternary tree expansion. New solutions and sign alternatives will be found, checked, rejected or added as new leaves.

Note that equalities can be handled in the same way as inequalities (consistently with the arbitrary choice of signs used for the sign-path). Thus, we shall not make the distinction with inequalities anymore. Once the TDT has been built, one thus disposes of a set E of linear inequalities, with which can be associated an oriented matroid.

12.3 Which mathematical results will be needed

Oriented matroid theory is a rich mathematical theory born in the 1970's as a combinatorial abstraction of linear algebra, convex geometry, and graph theory. In this paper, we focus on a small portion of this theory, that is, on vectorial oriented matroids and how they encode regions of all dimensions in real hyperplane arrangements. General oriented matroids are both combinatorial objects, by means of combinatorial axioms, and topological objects, by means of pseudosphere arrangements; see [21]. Section 12.3.1 is a short introduction to oriented matroids theory with some classical results that are necessary for our formal setting. Next, we will build on it for the sake of our model. We will give no technical details nor formal proofs for our constructions in this paper; see [22] for further deepening.

12.3.1 *Preliminaries on Oriented Matroid theory*

General setting. Let us start with a set of n linear inequalities in a vector space or an affine space (see Sec. 12.2). In the latter case, in a standard way, we embed the affine space into a vector space with one more dimension, corresponding to one more variable. The affine space corresponds to this variable having a value equal to 1. And we add the inequality (considered as being *at infinity*) where the additional variable is greater than zero. From now on, we thus assume that we are given a set of n inequalities in a real vector space (possibly including one inequality at infinity). Denoting the variables by x_i, $1 \leq i \leq r$, each inequality $\sum_{1 \leq i \leq r} \alpha_i.x_i \geq 0$ can be seen as a vector $(\alpha_1, \ldots, \alpha_r)$ of the ambient space. Up to considering a smaller ambient space, we can assume that these n vectors span the ambient space $\mathbb{R}^r$, and we will always make this assumption, which is crucial for further results and geometric representations.

We also assume that none of these vectors is the null vector. Hence, each induced linear equation $\sum_{1 \leq i \leq r} \alpha_i.x_i = 0$, yields an hyperplane of the ambient space (that is, a $(r-1)$-dimensional subspace), which is the set of vectors $(x_1, \ldots, x_r)$ satisfying the equation.

Combinatorial encoding in terms of oriented matroids. Accordingly to the previous setting, one disposes of a finite ground set $E = \{e_1, ..., e_n\}$ of a set of (non-null) vectors of the real space $\mathbb{R}^r$ (spanning the ambient space), possibly with repetitions of the same vector with different indices. Each vector $e \in E$ provides the equation $e.x = 0$, for $x \in \mathbb{R}^r$,

of a hyperplane H_e of $\mathbb{R}^r$ called *the hyperplane of* e, and provides the inequalities of two half-spaces $e.x > 0$ and $e.x < 0$, for $x \in \mathbb{R}^r$, called *the positive and negative half-space of* e, denoted H_e^+ and H_e^-, respectively. The hyperplane H_e can be equally denoted H_e^0. Also, the element at infinity will be denoted p, when it exists. Such a set of hyperplanes is called an *hyperplane arrangement.*

Those hyperplanes H_e and half-spaces H_e^+ and H_e^-, for all $e \in E$, subdivide the ambient space $\mathbb{R}^r$ into cells of various dimensions. For the sake of a geometric representation of these cells, notably used in the figures of this section, we actually consider a central unit sphere S^{r-1} of $\mathbb{R}^r$ as the ambient object, and the spheres S^{r-2} in S^{r-1} which are the intersections of the hyperplanes with S^{r-1}. When there is an element at infinity p, then, by symmetry, we can consider only the half-sphere S^{r-1} on the positive side of p, and we represent p as a sphere at infinity bounding the figure. In this way, the cells of $\mathbb{R}^r$ defined by the hyperplanes and half-spaces canonically correspond to cells of S^{r-1}, except the null vector of $\mathbb{R}^r$ which is not represented. See Fig. 12.4.

A *signed-set* C on E is defined by giving a sign in $\{+, -, 0\}$ to each element of E. The subset of elements with a $+$, $-$, or 0 sign is denoted by C^+, C^- or C^0, respectively. Thus, a signed-set yields a partition $E = C^+ \cup C^- \cup C^0$. The sign of $e \in E$ in the signed-set C is denoted C_e. The subset $\underline{C} = C^+ \cup C^-$ of E is called the *support* of C. For $A \subseteq E$, the signed-set $C \setminus A$ on $E \setminus A$ is defined to have $C^+ \setminus A$ as a positive part and $C^- \setminus A$ as a negative part. For example, writing elements of $E = \{1, \ldots, 5\}_<$ in a list, $C = [\texttt{---0+}]$ is the signed-set with $C^+ = \{5\}$, $C^- = \{1, 2, 3\}$ and $C^0 = \{4\}$.

For $x \in \mathbb{R}^r$, we define the *covector* C *associated with* x as the signed-set defined, for each $e \in E$, by giving a 0, $+$ or $-$ sign to e whether x belongs to the hyperplane, the positive half-space or the negative half-space of e, respectively. Formally, we have:

$$C_e = \text{sign}(e.x).$$

The (finite) set of all covectors associated with all $x \in \mathbb{R}^r$ is denoted by $\mathcal{C}\text{ov}$ and forms the family of covectors of an *oriented matroid* M on E. Formally, an oriented matroid is a finite ground set E provided with a family of signed-sets on E called its *covectors* and satisfying some combinatorial axioms. Here, the oriented matroid is called *vectorial* as it was built from vectors. Actually, an oriented matroid can be characterized by different families of signed-sets with different axiom systems, such as its cocircuits

or its topes addressed below. As E spans a vector space of dimension r, we say that the *rank* of M is r. The cells of the hyperplane arrangement are in canonical bijection with the covectors. Covectors combinatorially describe the relative positions of the cells. We call *tope* a maximal covector, that is, a covector not containing the zero sign. Topes correspond to *full-dimensional cells* of the space delimited by the hyperplanes of E (that is, to connected components of the complementary set in $\mathbb{R}^r$ of the union of the hyperplanes). We call *cocircuit* a minimal non-null covector. Geometrically speaking, cocircuits correspond to 1-dimensional cells. In a sphere representation, they correspond to the *points* obtained as intersections of spheres representing the elements of E.

The notions described above are illustrated in Fig. 12.4. For instance, covectors with no - sign correspond to cells bounding the cell encoded by [+++++] (see [0++++], [++++0], [+++0+], [0+++0], [+++00], and [0+00+]), topes with one - sign correspond to full-dimensional cells reached from the previous cell by crossing one hyperplane (see [-++++] and [+++-+]).

Conformal composition. Two covectors $C, D \in \mathcal{C}\text{ov}$ are called *conformal* (to each other) if $C^+ \cap D^- = C^- \cap D^+ = \emptyset$, that is, if no element $e \in E$ has a different non-zero sign in C and D. In other words, the corresponding cells are in the boundary of the same full-dimensional cell.

Given two conformal covectors C and D, the covector obtained by *conformal composition of C and D* is the covector denoted $C \circ D$, or $D \circ C$, whose positive elements are $C^+ \cup D^+$ and whose negative elements are $C^- \cup D^-$. Geometrically, when the corresponding cells are not comparable for inclusion, $C \circ D$ corresponds to the cell given by the strict convex hull of the cells corresponding to C and D. The next result will be crucial for us.

Theorem 12.1. *Every covector is the result of the conformal composition of the set of cocircuits that are conformal to it.*

The geometric interpretation in terms of full-dimensional cells is simply that every open bounded convex polytope (tope) is the strict convex hull (conformal composition) of its extremal points (cocircuits conformal to the tope). So, the above property can be seen as the combinatorial counterpart of this classical property in convex geometry. For instance, in Fig. 12.4, we have that [+++++] = [0+00+] ∘ [+++00] ∘ [0+++0], and we have that [-+0++] = [0+00+] ∘ [-+0+0].

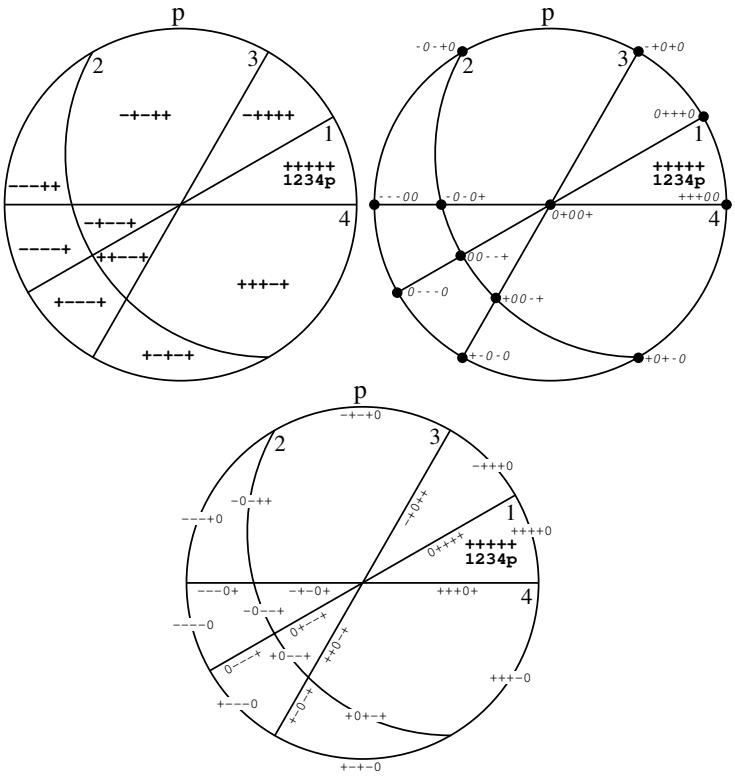


Fig. 12.4 Oriented matroid covectors encoding cells of the hyperplane arrangement, in a sphere representation with the element p at infinity. The signed-sets are indicated as lists with respect to the ordering $E = \{1, 2, 3, 4, p\}_<$. The positive (and negative) sides of the hyperplanes are indicated by the full-dimensional cell whose signs [+++++] are all positive. Topes are represented in the upper left figure. Cocircuits are represented in the upper right figure. Other non-null covectors are represented in the third figure. This oriented matroid is obtained from the equations of the running example (Sec. 12.2.2).

Minors. Given a subset A of E, we define the oriented matroid $M\backslash A$ obtained by *deletion of A from M* as the oriented matroid defined as above from the vectors in $E \setminus A$ (its covectors can be characterized in a combinatorial way, but we omit this). Observe that a representation of M directly yields a representation of $M\backslash A$. An example of deletion is shown in the left side of Fig. 12.5.

Given a subset A of E, we define the oriented matroid M/A obtained by *contraction of A from M* in the following way (again a combinatorial characterization exist that we omit). Geometrically, the contraction of A

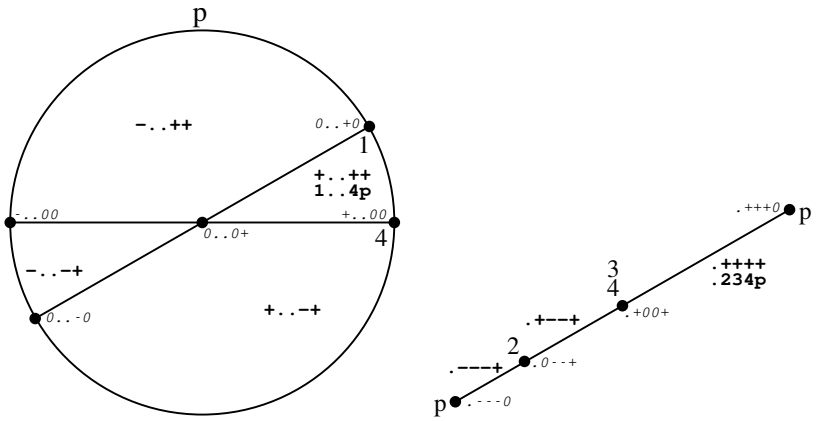


Fig. 12.5 Oriented matroid minors. Spherical representations of the oriented matroid $M\backslash 23$ on the left and of the oriented matroid $M/1$ on the right, where M is the oriented matroid of Fig. 12.4. Their topes and cocircuits are indicated (dots stand for omitted elements).

consists in considering only the set of cells contained in every hyperplane in A. Precisely, we consider the subspace $H_A = \bigcap_{a\in A} H_a$ as a new ambient space, and we consider the arrangement of hyperplanes $\{H_e \cap H_A \mid e \in E \setminus A,\ H_e \not\supseteq H_A\}$ indexed by $E \setminus A$ in this space (the elements e such that $H_e \supseteq H_A$ become so-called loops, which encode the null vector and are omitted in the structure). In this way, a representation of M directly contains a representation of M/A. An example of contraction is shown in the right side of Fig. 12.5.

The oriented matroids of type $M\backslash A/A'$ for $A \subseteq E$ and $A' \subseteq E$ with $A \cap A' = \emptyset$ are called *minors* of M. The ordering of the deletion/contraction operations does not matter since $M\backslash A/A' = M/A'\backslash A$.

As noted above, a representation of M directly yields a representation of $M\backslash A$. However, the dimension of the space spanned by $E \setminus A$ is not necessarily the same as the initial ambient space spanned by E, which was our assumption (that is, the rank of $M\backslash A$ can be smaller than the rank of M). In this case, removing elements from a spherical representation of M does not yield a proper spherical representation of $M\backslash A$ under the same assumption, and points in the resulting representation do not correspond to cocircuits anymore. This subtlety will be important for us. The next lemma states that cocircuits of minors of M can be seen as cocircuits of M as soon as the assumption is preserved.

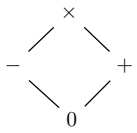
Lemma/Definition 12.2. Consider the minor $M' = M\backslash A/A'$ of M. Assume that the rank of $M\backslash A$ is equal to the rank of M. Then, for any cocircuit C' of M', there exists a unique cocircuit C of M such that $C' \setminus A = C$. We call C the *lift* of C' in M.

For example, consider the cocircuit [0..-0] of $M\backslash 23$ in Fig. 12.5. Its lift in M is the cocircuit [0---0] of M in Fig. 12.4.

12.3.2 *Encoding regions with super-covectors*

What follows does not belong to standard oriented matroid theory. We introduce original concepts for our application.

Let us define $\mathcal{Q} = \{\ 0,\ -,\ +,\ \times\ \}$ as the set of *super-signs*, where $\times$ is called *the undetermined sign*. The set $\mathcal{Q}$ is ordered with $0 \leq + \leq \times$ and $0 \leq - \leq \times$, forming a lattice (depicted below), whose *join* and *meet* operators are denoted by $\vee$ and $\wedge$, respectively:



Definition 12.3. Let E be a finite set. A *super-signed-set* over E (*sss* for brevity), is an element $U \in \mathcal{Q}^E$. We naturally extend, componentwise, the lattice structure of $\mathcal{Q}$ to a lattice structure of $\mathcal{Q}^E$. Let $U, V \in \mathcal{Q}^E$. We say that U is *smaller* than V, noted $U \preceq V$, if $U_e \leq V_e$, for all $e \in E$. Finally, $U \vee V$ and $U \wedge V$, are defined by $(U \vee V)_e = U_e \vee V_e$ and $(U \wedge V)_e = U_e \wedge V_e$, respectively, for all $e \in E$.

For $e \in E$, the super-sign associated to e in U is denoted by U_e. For a super-sign $\sigma \in \mathcal{Q}$, we define the set $U^\sigma = \{e \in E : U_e = \sigma\}$. Observe that if $U^\times = \emptyset$, then U is a signed-set, as addressed in Sec. 12.3.1.

Observe that two signed-sets U and V are conformal (Sec. 12.3.1) if and only if no element of $U \vee V$ has the undetermined sign. In that case, we have $U \vee V = U \circ V$. For example, we have [+0] $\vee$ [-0] = [x0], [+0] $\vee$ [0-] = [+0] $\circ$ [0-] = [+-], [+x] $\vee$ [00] = [+x], and [+x] $\vee$ [x0] = [xx]. Observe also that, for two signed-sets U and V, if $U \preceq V$ then U and V are conformal.

Let M be an oriented matroid on E as built in Sec. 12.3.1. Recall that each $e \in E$ is associated with one equation that partition the space into one hyperplane H_e^0 and two (open) halfspaces H_e^+ and H_e^-.

Property 12.1. For a sss U over E, the following conditions are equivalent:

- the following region is non-empty:
$$\text{region}(U) = \bigcap_{\substack{e\in E \\ U_e\neq\times}} H_e^{U_e}$$
- the signed-set $U\setminus\big(U^\times\cup U^0\big)$ on $E\setminus\big(U^\times\cup U^0\big)$ is a maximal covector (or tope) of the minor $M\backslash U^\times/U^0$ of M.

Definition 12.4. A sss U satisfying the conditions of Property 12.1 is called a *super-covector* of M, or *scovector* for brevity. Also, region(U) is called the *region of* U; and $\overline{\text{region}}(U)$ denotes the topological closure of region(U). Two super-covectors U and V representing the same region are considered as *equivalent* and noted $U\sim V$. The family of all scovectors of M will be denoted by $\mathcal{SC}\text{ov}$.

We call *null-scovector* the scovector with only `0` signs. Note that its region is $\{0\}$, consisting of the null-vector only. Note also that the scovector with only $\times$ signs yields the whole ambient space as a region.

12.3.3 *Super-covectors versus covectors*

Super-covectors essentially consist of groups of covectors/regions induced by covectors/regions of minors. Obviously, covectors of M are also scovectors. If C is a covector, then region(C) is the cell of C as addressed in the previous section. The scovectors with no 0 sign and no $\times$ sign are the conventional topes (or maximal covectors) of the oriented matroid (Sec. 12.3.1). The scovectors with no 0 sign are called *super-topes* of M. We mention that this notion was addressed under the same name in [23] and is equivalent to the so-called notion of T-convex sets; see [21, Sec. 4.2].

As an example, Fig. 12.6 shows various scovectors of the oriented matroid of Fig. 12.4. Observe how different scovectors may represent the same region. For instance, in Fig. 12.6: the scovectors [`+++0+`], [`+xx0+`] and [`xx+0+`] represent the same region, which is the cell of the covector [`+++0+`] of M (and the cell of the covector [`+++.+`] of the minor $M/4$); the scovectors [`0x--+`], [`0x-x+`] and [`0xx-+`] represent the same region, which is also the cell of the covector [`0.--+`] of the minor $M\backslash 2$ and the cell of the covector [`..--+`] of the minor $M/1\backslash 2$.

Definition 12.5. For two scovectors U and V, let us denote $U\trianglelefteq V$ if, for all $e\in E$, we have $U_e=V_e$ or $V_e=\times$. Equivalently: $U\trianglelefteq V$ if and only if $U\preceq V$ and $U^0\subseteq V^0\cup V^\times$.

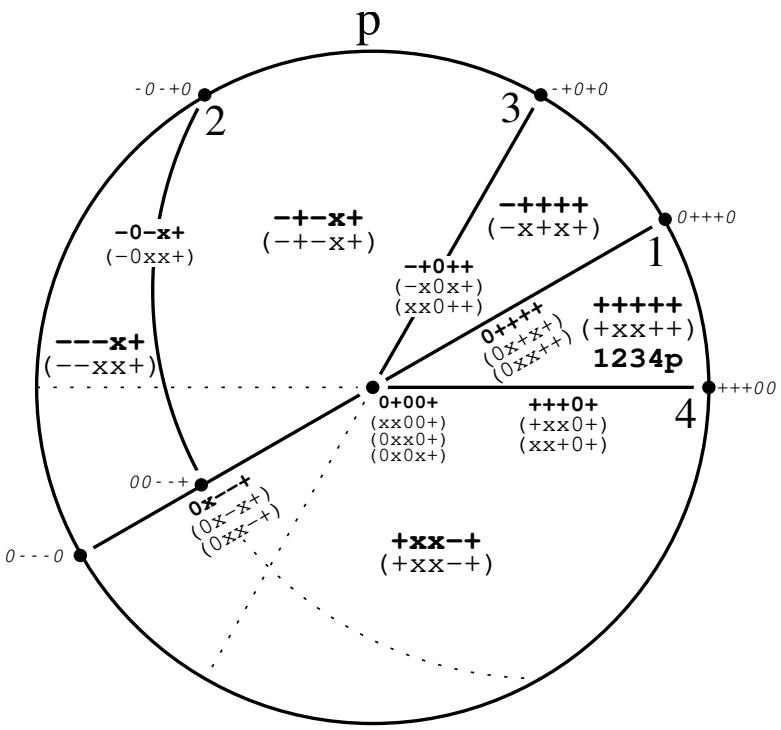


Fig. 12.6 The figure shows various super-covectors (or scovectors) for the oriented matroid of Fig. 12.4. Each scovector is written in its region (Definition 12.4). The bold scovector is the representative of the region (Proposition/Definition 12.7), and some equivalent scovectors are written below. The dotted portions of elements represent those which are signed $\times$ for the regions that they cross. The cocircuits, that will serve as borders of scovectors, are written in italics (Sec. 12.3.6). This set of scovectors forms a fragmentation of the oriented matroid, that is, their regions form a partition of the sphere (Definition 12.6). This fragmentation is obtained from the toy example addressed in Sec. 12.2.2 and Fig. 12.2.

As seen below in Property 12.3, this $\trianglelefteq$ relation corresponds to an inclusion relation for regions of scovectors. Observe that two covectors are not comparable for $\trianglelefteq$ unless they are equal. The $\trianglelefteq$ relation will be useful for handling scovectors representatives as defined in Sec. 12.3.4.

Property 12.2. For a scovector U, we have:

- covectors $C \trianglelefteq U$ are obtained from U by possibly replacing every $\times$ sign in U with a sign in $\{+, -, 0\}$, and keeping only covectors of M;

- covectors $C \preceq U$ are obtained from U by possibly replacing every $\times$ sign in U with a sign in $\{+,-,0\}$ and/or every $\{+,-\}$ sign in U with a 0 sign, and keeping only covectors of M;
- $\mathrm{region}(U) = \bigsqcup_{\substack{C \in \mathcal{C}\mathrm{ov} \\ C \trianglelefteq U}} \mathrm{region}(C)$ and $\overline{\mathrm{region}}(U) = \bigsqcup_{\substack{C \in \mathcal{C}\mathrm{ov} \\ C \trianglelefteq U}} \mathrm{region}(C)$ (where $\sqcup$ denotes a disjoint union).

For instance, in Fig. 12.6, the scovector [+xx−+] is the union of the following covectors from Fig. 12.4: [+−−−+], [++−−+], [+−+−+], [+++−+] (topes), [+0−−+], [+−0−+], [++0−+], [+0+−+] (intermediate covectors) and [+00−+] (cocircuit).

Definition 12.6. A *fragmentation* of M is a set $\mathcal{F}$ of scovectors of M whose regions partition the ambient space (that is, regions are disjoint to each other and their union equals the whole space). Equivalently, in a combinatorial way: the sets $\big\{C \mid C \in \mathcal{C}\mathrm{ov},\ C \trianglelefteq U\big\}$, for $U \in \mathcal{F}$, partition the set of covectors of M. (This is a combinatorial counterpart of Definition 12.1.)

12.3.4 *Representative of a super-covector*

As noticed above, various super-covectors may represent the same region. The notion of representative will be essential to compare and relate homodromies. Representative scovectors are written in bold in Fig. 12.6.

Proposition/Definition 12.7. Let U be a scovector of M. There exists a unique scovector of M, which we denote by $\mathrm{rep}(U)$, such that $U \sim \mathrm{rep}(U)$ and $\mathrm{rep}(U)^{\times}$ has the smallest possible number of $\times$ signs. It satisfies

$$\mathrm{rep}(U) = \bigwedge_{V \sim U} V = \bigvee_{\substack{C \in \mathcal{C}\mathrm{ov} \\ C \trianglelefteq U}} C.$$

If the region of U is $\{0\}$ (consisting of the null-vector only), then $\mathrm{rep}(U)$ is the null-scovector (with only 0 signs). We call *representative* of U, or of $\mathrm{region}(U)$, the scovector $\mathrm{rep}(U)$. A scovector which is equal to its representative is called a *representative scovector*. Clearly, covectors are representative scovectors.

Property 12.3. For two scovectors U and V, we have the following:

(i) $U \sim V$ if and only if $\text{rep}(U) = \text{rep}(V)$.
(ii) If $U \preceq V$ then $\text{rep}(U) \preceq \text{rep}(V)$, and if $U \trianglelefteq V$ then $\text{rep}(U) \trianglelefteq \text{rep}(V)$.
(iii) $\text{region}(U) \subseteq \text{region}(V)$ if and only if $\text{rep}(U) \trianglelefteq \text{rep}(V)$.
(iv) $\overline{\text{region}}(U) \subseteq \overline{\text{region}}(V)$ if and only if $\text{rep}(U) \preceq \text{rep}(V)$.

For instance, in Fig. 12.6, the scovectors [+++0+], [+xx0+] and [xx+0+] admit the same representative scovector [+++0+], and the scovectors [0x--+], [0x-x+] and [0xx-+] admit the same representative scovector [0x--+].

Note that "rep" cannot be omitted in the above statements. For instance, in Fig. 12.6, the two scovectors [x+00+] $\sim$ [0+x0+] yield the same region (with representative [0+00+]) but are not comparable.

Note also that, for two scovectors U and V, $\text{rep}(U \vee V)$ can be different from $\text{rep}(U) \vee \text{rep}(V)$. (Thus, in general, U is not equivalent to $\bigvee_{V \sim U} V$.) For instance, [0x-x+] $\vee$ [0xx-+] = [0xxx+] whose region is the whole part of H_1 in the positive side of H_p, larger than the region of [0x--+].

However, with the above properties, one can see that the lattice structure of $\mathcal{Q}^E$ naturally induces a consistent lattice structure for the set of representative scovectors. (The join and meet operations between $\text{rep}(U)$ and $\text{rep}(V)$ are given by $\text{rep}(\text{rep}(U) \vee \text{rep}(V))$ and $\text{rep}(\text{rep}(U) \wedge \text{rep}(V))$, respectively.) This lattice is isomoprhic to a lattice for closures of scovector's regions.

Finally, representative scovectors can be considered as the useful scovectors for a practical encoding and handling of the regions. Practically, they can be computed using the construction of Sec. 12.3.6 below.

12.3.5 *Contiguity between super-covectors*

The notion of contiguity between homodromies will be central to use our model, and it can be directly deduced from the oriented matroid structure.

Property 12.4. Let U and V be two scovectors. Then $U \wedge V$ is a scovector.

We have $\overline{\text{region}}(U) \cap \overline{\text{region}}(V) \neq \{0\}$ (where 0 denotes the null-vector) if and only if the $\text{rep}(U \wedge V)$ is not the null scovector (with only 0 signs), that is, if and only if $\text{region}(U \wedge V) \neq \{0\}$.

Furthermore, if $\text{region}(U) \cap \text{region}(V) = \emptyset$ then the region of $U \wedge V$ is the greatest common face of the closures of the regions of U and V.

Definition 12.8. Two scovectors U and V with disjoint regions are called *contiguous* when the union of their regions is connected, that is, when one can pass continuously from $\text{region}(U)$ to $\text{region}(V)$.

Proposition 12.1. *For two scovectors U and V with disjoint regions, the following conditions are equivalent:*

(i) *U and V are contiguous;*
(ii) *$\overline{\text{region}}(U) \cap \text{region}(V)$ or $\text{region}(U) \cap \overline{\text{region}}(V)$ is non-empty;*
(iii) *$\text{rep}(U \wedge V)$ satisfies $\text{rep}(U \wedge V) \trianglelefteq \text{rep}(U)$ or $\text{rep}(U \wedge V) \trianglelefteq \text{rep}(V)$.*

Application. Further properties and underlying combinatorial structures can be obtained by exploiting, for instance, the approach of the above properties. Starting with the set of all representatives of scovectors of a fragmentation, and applying all possible $\wedge$ and $\vee$ operations, a lattice is naturally built. This lattice is a combinatorial representation of the way the algorithm acts on the input space and contains all the necessary information to perform qualitative reasoning and test data generation tasks. Such applications will be given in Sec. 12.4 using the above results as combinatorial criteria.

12.3.6 *Border of a super-covector*

Given a scovector U, our goal is here to compute $\text{rep}(U)$ by a join operation applied to a set of cocircuits of M, or of a suitable minor of M, that we call the *border* of U. This construction yields a way to compute representative scovectors, and, furthermore, to compute boundary test data in Sec. 12.4.3. Let us state a definition and result in the most general case, that we will explain and illustrate in the simpler most frequent cases. Various equivalent definitions are possible but will be omitted here.

Definition 12.9. Let U be a scovector of M. Let X be the union of all supports of cocircuits of M which are contained in $U^\times$. Let us denote $M' = M \backslash U^\times / U^0$ and $U' = U \setminus \left(U^\times \cup U^0\right)$. Consider the cocircuits of M' which are conformal to U'. Then, the *border of U in M* is the set of scovectors of M which are obtained by taking the lifts in $M \backslash X$ of these cocircuits of M', and then by adding a $\times$ sign to all elements in X. The border is denoted by $\partial_M(U)$, or by $\partial(U)$ for brevity.

Scholia. In Definition 12.9, in the case where $X = \emptyset$, which we call the *tame case*, $\partial(U)$ is a set of cocircuits of M. (In this case, equivalently, the rank of $M \backslash U^\times$ equals the rank of M, and it is the most frequent in applications.) In the general case, $\partial(U)$ is a set of very special scovectors

of M: for $V \in \partial(U)$, we have that $V^{\times} = X$ and $V \setminus X$ is a cocircuit of $M \setminus X$. Furthermore, $\partial_{M\setminus X}(U \setminus X) = \{V \setminus X \mid V \in \partial_M(U)\}$.

Theorem 12.2. *Let U be a scovector of M. Then* $\mathrm{rep}(U)$ *is the join of its border:*

$$\mathrm{rep}(U) = \bigvee_{C \in \partial(U)} C.$$

Furthermore, for two scovectors U and V, we have $U \sim V$ if and only if $\partial(U) = \partial(V)$.

First, let us consider the particular case where $U^{\times} = \emptyset$ (hence $X = \emptyset$) and $U^0 = \emptyset$. In this case, $U = U'$ is a maximal covector of $M = M'$, and $\partial(U)$ is simply the set of cocircuits of M which are conformal to U. Then, the above result can be written $\mathrm{rep}(U) = U = \bigcirc_{C\in\partial(U)} C = \bigvee_{C\in\partial(U)} C$, and it is essentially a reformulation of Theorem 12.1, which is given here by the central equality. (Obviously, $U = \mathrm{rep}(U)$ as it has no $\times$ sign, and the conformal composition can be replaced with the join operation as the two operations coincide for signed-sets which are conformal to each other, as observed in Sec. 12.3.2.) As mentioned in Sec. 12.3.1, the geometric interpretation of the above result is the following classical result in geometry: a bounded convex polytope (encoded by a covector) is the convex hull (encoded by the composition operation) of its extremal points (encoded by conformal cocircuits). In the case where $U^{\times} = \emptyset$ and, possibly, $U^0 \neq \emptyset$, we have exactly the same result and interpretation but in the minor $M' = M/U^0$, and the border of U is also formed by cocircuits of M.

For instance, in Fig. 12.6, for the maximal covector $U = [+++++]$, the border $\partial(U)$ is formed by $[0{+}00{+}]$, $[0{+}{+}{+}0]$ and $[{+}{+}{+}00]$, whose join equals U; and for the covector $V = [-{+}0{+}{+}]$, the border $\partial(V)$ is formed by $[0{+}00{+}]$ and $[-{+}0{+}0]$, whose join equals V.

Second, consider now the case where $X = \emptyset$ (but, possibly, $U^{\times} \neq \emptyset$). In this case, we call U a *tame* scovector, and the border of U is also formed by cocircuits of M. By Property 12.1, the signed-set $U' = U \setminus \left(U^{\times} \cup U^0\right)$ is a maximal covector of $M' = M\setminus U^{\times}/U^0$. Since $X = \emptyset$, then the rank of $M\setminus U^{\times}$ equals the rank of M (this is a lemma which we omit). Hence, the lift notion from Lemma/Definition 12.2 is well-defined in M'. Consider the border $\partial_{M'}(U')$ of U' in M' as defined in the case above. Then the cocircuits in $\partial_M(U)$ are the lifts in M of cocircuits of M' belonging to $\partial_{M'}(U')$.

As discussed in Sec. 12.3.1, being tame means that the region of U in the ambient space of M and the region of U' in the ambient space of M' coincide. According to our experiments on program testing, most (or all) scovectors are tame in applications. For instance, all scovectors depicted in Fig. 12.6 are tame. The border of the scovector $U = [\texttt{+xx-+}]$ is formed by the cocircuits [0---0], [0+00+] and [+++00] whose join equals U. The border of the scovector [-0-x+] is formed by the cocircuits [-0-+0] and [00--+].

Third, in the non-tame case, scovectors of the border are no more cocircuits of M. By properties of the rank function, we have that $r(M\backslash X) = r(M\backslash U^{\times}) = r(M) - r(X)$. Hence the lifting notion is well-defined in $M\backslash X$. By Property 12.1, the signed-set $U' = U \setminus \left(U^{\times} \cup U^{0}\right)$ is a maximal covector of $M' = M\backslash U^{\times}/U^{0}$. Consider the border $\partial_{M'}(U')$ of U' in M' as defined in the first case. Then, the scovectors in $\partial_{M}(U)$ are obtained from $\partial_{M'}(U')$ as said in Definition 12.9 by using Lemma/Definition 12.2 in $M\backslash X$.

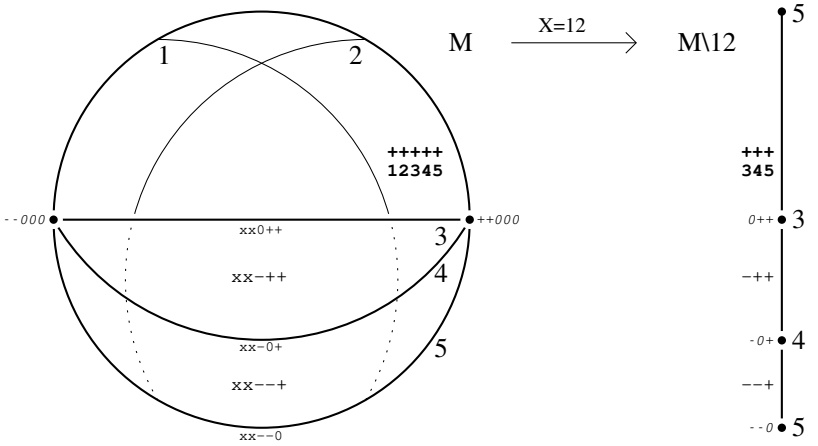


Fig. 12.7 Example of non-tame scovectors and their borders. The positive half-spaces are indicated by the region of the [+++++] covector. For any of the five depicted scovectors with $U^{\times} = 12$, we have that $U^{\times}$ contains the support $X = 12$ of the cocircuit [++000], hence these scovectors are non-tame. The spherical representation of $M\backslash X$ induced by the spherical representation of M in the sphere S^2 (on the left, with 5 at infinity) is thus not proper (the cocircuits do not correspond to points anymore, as shows the point associated with [++000] in M, which would be associated with [000] in $M\backslash X$ but which is not a cocircuit). The proper spherical representation of $M\backslash X$ is given in the sphere S^1 (on the right, with 5 still at infinity). The five above scovectors yield covectors of $M\backslash X$, as shown on its proper representation. The border of [..-++] in $M\backslash X$ is formed by [..0++] and [..-0+]. Thus, the border of [xx-++] in M is formed by [xx0++] and [xx-0+]. Observe on the left picture how the two corresponding half-linear-subspaces (of dimension 2 in $\mathbb{R}^3$) delimit the region of [xx-++].

An example of non-tame scovectors and their borders is given and explained in Fig. 12.7. Note that the scovectors belonging to a border are either cocircuits (tame case) or very special scovectors (cf. Scholia above). When they correspond to cocircuits of a minor $M\backslash X$ with a loss of rank, they are supported by half-linear-subspaces of the initial ambient vector space (which are just half-lines in the tame case).

Application. The above definition and result have their numeric counterpart that will be used for limit test data generation (see in Sec. 12.4.3). Each cocircuit of $\partial(U)$ can be realized as a real vector (recall that it corresponds a precise point in a spherical representation). In general, one can compute a real vector belonging to the region of each scovector of $\partial(U)$. Then, performing a join operation on the combinatorial structure amounts to perform a convex combination of the associated real vectors. In this way, by Theorem 12.2, one can generate real vectors on the extreme boundary faces — encoded by $\partial(U)$ — of the region of U, and in their neighbourhood inside the region of U.

12.4 How the model is used

Once the model (consisting of real inequalities and combinatorial information) has been computed (Sec. 12.2), one can determine the underlying oriented matroid which vehiculates additional combinatorial information such as representative scovectors and borders of homodromies (Sec. 12.3). We present in this section the three "outputs" of the qualitative model that one can derive from the previous computations: envision graphs, order-magnitude reasoning, and boundary test data generation. Our examples will still be based on our `WaterPhases` algorithm.

12.4.1 *Relations between Homodromies and the Envision Graph*

Intuitively speaking, an envision graph is a qualitative visualisation of the behaviour of the algorithm.

Homodromies form a partition of the ambient input space (called a *fragmentation* in Definitions 12.1 and 12.6). Moreover, homodromies, as regions of scovectors, are bijectively encoded in terms of their representative scovectors (Sec. 12.3.4). It is then natural to use the contiguity property of Proposition 12.1 as a straightforward combinatorial criterion for building the following adjacency graph.

Definition 12.10. Let us consider a *fragmentation* $\mathcal{F}$. The *homodromy graph* of $\mathcal{F}$ is the undirected graph G whose vertices are elements of $\mathcal{F}$ and whose edges are pairs of contiguous scovectors in $\mathcal{F}$.

In practical applications, homodromy graphs are relatively dense graphs, since they express the geometric configuration of all the feasible paths of the algorithm. It is then tempting to aggregate neighbor vertices with respect to a user-defined equivalence relation $\simeq$. For instance, distinct homodromies can correspond to the same output of the initial software. In our example, $\simeq$ will be "having the same output" (see Figs. 12.8 and 12.9). In other cases, the equivalence relation can be established using the subset of inputs that are numerically influencing outputs (see example below and Fig. 12.10). More formally:

Definition 12.11. Let G be the homodromy graph of the fragmentation $\mathcal{F}$, and let $\simeq$ be an equivalence relation over $\mathcal{F}$. The *envision graph* of $\mathcal{F}$ with respect to $\simeq$ is the graph obtained from G by contracting edges with equivalent endpoints. The resulting vertices of G' are consistently labelled with the equivalence classes of $\simeq$.

In Fig. 12.8 is given the homodromy graph of Fig. 12.6. Its corresponding envision graph (where homodromies having the same output are aggregated) is illustrated in Fig. 12.8 and refined in Fig. 12.9. As illustrated in this example, the envision graph can also be refined as a directed graph in order to take into account the influence of increasing/decreasing variable changes.

As another example, let us consider the following source code for the algorithm $\texttt{MAX}_N$ which computes the maximum of an array of N elements.

```
max = a[0]
for i in range(1, N):
   if a[i] > max: max = a[i]
return max
```

In Fig. 12.10, are given the envision graphs of the $\texttt{MAX}_3$ and $\texttt{MAX}_4$, for the equivalence relation $\simeq$ defined as follows: two homodromies are equivalent if their output is influenced by the same input variable. This information is automatically collected during the TDT construction (Sec. 12.2.3). More generally, the algorithm $\texttt{MAX}_N$ admits 3^{N-1} homodromies (feasible paths) and its underlying oriented matroid is a classical one known as the *braid arrangement* (of dimension $N-1$, admitting $N!$ full-dimensional regions, in

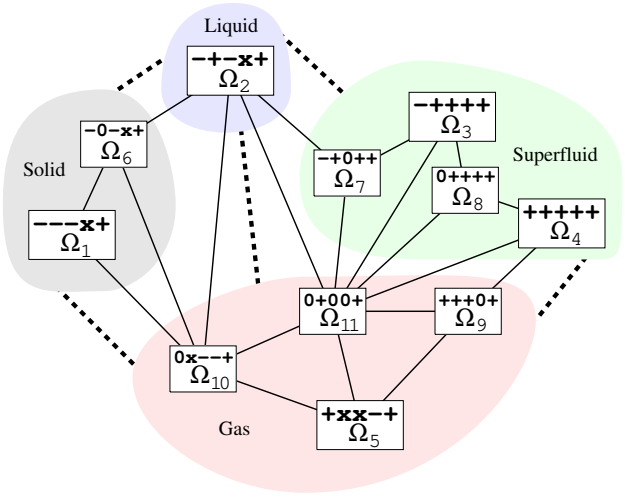


Fig. 12.8 Homodromy graph for the fragmentation of Fig. 12.6: vertices are the scovectors of Fig. 12.6, and edges reflect the contiguity relations between regions in Fig. 12.6. Envision graph for this fragmentation with respect to the output of the `WaterPhases` algorithm of Sec. 12.2.2 and Fig. 12.2: vertices are groups of scovectors corresponding to the same output, edges (represented by dashed segments) are induced by edges of the homodromy graph between those groups. Note that each homodromy is given with its representative scovector (Sec. 12.3.4), different from its initial scovector (Table 12.1).

bijection with permutations, or acyclic orientations of the complete graph). It contains $2^N - 2$ cocircuits. Among them, $2N - 1$ are sufficient to express combinatorially all the homodromies by means of their borders (Sec. 12.3.6). The envision graph of $\texttt{MAX}_N$, according to $\simeq$, turns out to be the complete graph K_N, which we refine in the following way: to any given input a_i corresponds a node labelled a_i (expressing the fact that the a_i input is influencing the output), admitting $N - 1$ outgoing arrows a_j, $j \neq i$, and $N - 1$ ingoing arrows, all equal to a_i.

Envision graphs are mainly destinated to the visualisation of local behaviours. When equations are linear, it is possible to have a global vision. However, when equations become very numerous, the envision graph can be very complex to visualize globally. It is then preferable to use it as a simulation tool (the simulation aspect is not in the scope of this paper).

Let us end with an application of the intersection criterion of Property 12.4. Suppose that one wishes to know what happens when three different phases meet at a common boundary surface. Using the four phases

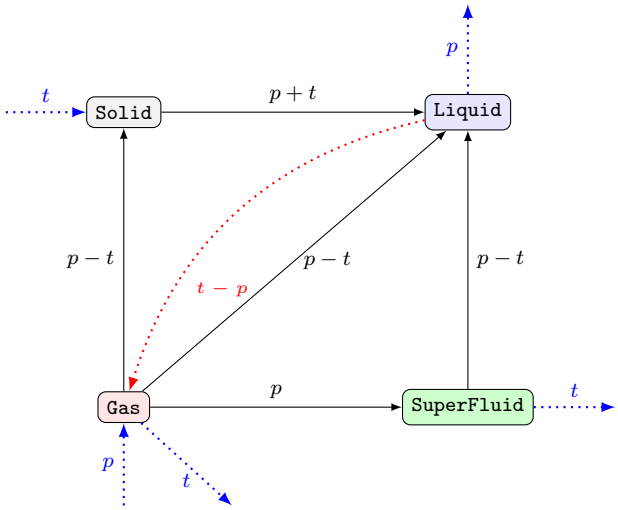


Fig. 12.9 Refined envision graph of the `WaterPhases` algorithm, automatically generated by our tool (with some minor cosmetic changes). The underlying graph is the same as the one given in Fig. 12.8. Labels p and t stand for *pressure* and *temperature*. The node `SuperFluid` aggregates the 4 contiguous `SuperFluid` homodromies Ω_3, Ω_4, Ω_7, Ω_8. The arrow from `Gas` to `Liquid` labeled $p-t$ expresses that *pressure* must *increase* and/or the temperature must decrease to allow the transition. Arrows are reversible: the previous arrow could be reversed from `Liquid` to `Gas` and labeled $t-p$ (dotted arrow). Arrows with no predecessors/successors (infinity arrows) mean that any increase/decrease will not change the phase. For instance, in the `Gas` phase, any increase of the *temperature* and any decrease of *pressure* will not change the phase; equally, only a *pressure* increase will allow the transition to `SuperFluid` phase. Note also that there is no possible direct transition from `Solid` to `SuperFluid`.

homodromies of maximal dimension, Ω_1(`Solid`), Ω_2(`Liquid`), Ω_3(`SuperFluid`), Ω_5(`Gas`) and this criterion, one directly gets the possible triple points:

- $\text{rep}(\Omega_1) \wedge \text{rep}(\Omega_2) \wedge \text{rep}(\Omega_3) = [\texttt{---x+}] \wedge [\texttt{-+-x+}] \wedge [\texttt{-++++}] = [\texttt{-00++}] \sim [\texttt{00000}]$ which corresponds to the null vector, which does not belong to the affine initial input space (outside the specifications). Hence, there is no possible triple point between the `Solid`, `Liquid` and `SuperFluid` phases.
- $\text{rep}(\Omega_1) \wedge \text{rep}(\Omega_2) \wedge \text{rep}(\Omega_5) = [\texttt{---x+}] \wedge [\texttt{-+-x+}] \wedge [\texttt{+xx-+}] = [\texttt{00--+}] = \text{rep}(\gamma_1)$
- $\text{rep}(\Omega_2) \wedge \text{rep}(\Omega_3) \wedge \text{rep}(\Omega_5) = [\texttt{-+-x+}] \wedge [\texttt{-++++}] \wedge [\texttt{+xx-+}] = [\texttt{0+00+}] = \text{rep}(\gamma_2)$

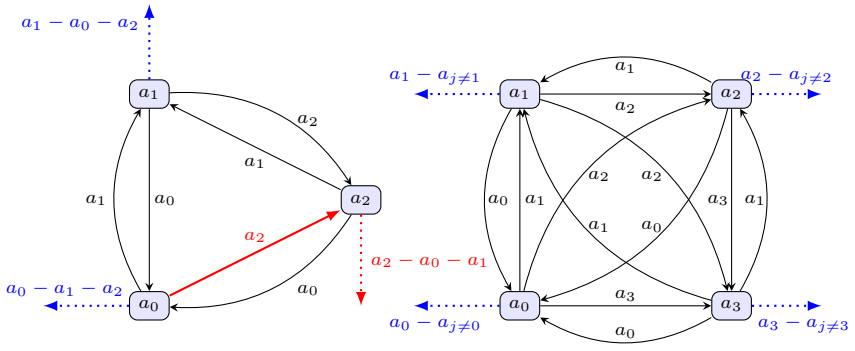


Fig. 12.10 The Envision Graphs of $\texttt{MAX}_3$ (left) and $\texttt{MAX}_4$ (right) algorithms, refined with arrows. Nodes are automatically labeled, according to the (here unique) input influencing the result. The arrow from a_0 to a_2, labeled a_2 means that when the maximum is the first element of the array (a_0), and the third element (a_2) is incremented then, eventually, a_2 will be selected as the maximum. If a_2 "continues" to increase or any other element decreases, the algorithm will stay at the same state a_2 (infinite arrow labeled $a_2 - a_0 - a_1$). Each of the 4 nodes of $\texttt{MAX}_4$ graph represents a 4-dimensional polytope. (One can note that reversing the arrows, yields the envision graphs of similar $\texttt{MIN}_3$ and $\texttt{MIN}_4$ algorithms.)

where γ_1 and γ_2 are indicated in Figs. 12.2 and 12.11 (we use representatives of homodromies, shown in Figs. 12.6 and 12.8, possibly different from their initial scovectors from Table 12.1).

12.4.2 *Order-of-Magnitude Reasoning and Qualitative Execution*

We introduce in this section an original order-of-magnitude algebra which can be considered as a non-standard version of an interval based order of magnitude algebra from [16]. We then illustrate how this formalism can be consistently combined with the combinatorial spatial properties of scovectors, to obtain useful qualitative conclusions.

Order-of-Magnitude Reasoning. Non-standard analysis handles hyperreal numbers, that is, infinitesimally small numbers (called *infinitesimals*) and infinitesimally big (called *infinite*) numbers, with an equivalence relation, noted $\approx$ meaning "infinitesimally close"; see [24,25]. In our model, ϵ stands for any strictly positive infinitesimal, d for any standard strictly positive finite (but not infinitesimal) real number, and ω stands for any positive infinite number (and their opposites are represented using a $-$ sign).

The qualitative value associated to an input of the algorithm, say p, will be denoted $[p]$ and will be a closed interval. For instance $[p] \approx [\epsilon, \omega]$ stands for a strictly positive input p with an unknown order-of-magnitude. The interval $[-\omega, \omega]$ expresses undeterminacy; $[-d, \epsilon]$ stands for any finite real (possibly negative) that is inferior that any standard positive real number. The four interval arithmetic operations $(\oplus, \otimes, \ominus, \oslash)$ constitute an interval extension of the hyperreal operations. Some examples: $[d, d] \oplus [\epsilon, \epsilon] \approx [d, d]$; $[d, d] \ominus [\omega, \omega] \approx [-\omega, -\omega]$, $[\epsilon, \epsilon] \otimes [\omega, \omega] \approx [\epsilon, \omega]$; $[-d, \epsilon] \oplus [0, d] \approx [-d, d]$; $[-d, -d] \otimes [-\epsilon, \epsilon] \approx [-\epsilon, \epsilon]$.

Qualitative execution. Suppose in the following that, in the toy algorithm, the temperature is a very small number, of unknown sign ($[t] \approx [-\epsilon, \epsilon]$) and the pressure has a standard positive value ($[p] \approx [d, d]$). Using linearity of $e_2 = 10t+p$ and the neutrality of $[d, d]$, it is then straightforward to evaluate the qualitative value of e_2. In fact, all coefficients, considered as standard reals, ($[d, d]$) can be dropped and we obtain: $[e_2] \approx [10 \cdot t + 1 \cdot p] \approx [10] \otimes [t] \oplus [1] \otimes [p] \approx [d, d] \otimes [t] \oplus [d, d] \otimes [p] \approx [t] \oplus [p] \approx [-\epsilon, \epsilon] \oplus [d, d] \approx [d, d]$. We conclude that the equation e_2 will be positive. For the other equations one gets:

- $[e_1] \approx [t] - [p] \approx [-\epsilon, \epsilon] \ominus [d, d] \approx [-d, -d]$ thus its sign is $-$
- $[e_3] \approx [t] - [p] - [d, d] \approx [-\epsilon, \epsilon] \ominus [d, d] \ominus [d, d] \approx [-d, -d]$, with sign $-$
- $[e_4] \approx [p] - [d, d] \approx [d, d] \ominus [d, d] \approx [-d, d]$ thus its sign is unknown, $\times$

Grouping all equations' signs, and adding a + sign at the end (as before for the equation at infinity) we obtain the scovector $U = [\texttt{-+-x+}]$ which corresponds to the `Liquid` homodromy Ω_2. *We conclude that, when temperature is very close to zero and pressure has a normal (non-negligible) positive value, the result will be the* `Liquid` *phase.*

Let's take another example: *what happens when the temperature is very high and pressure is normal (non-negligible) but unknown?* We have $[t] \approx [\omega, \omega]$ and $[p] \approx [-d, d]$. Evaluating, as previously, equations signs, we obtain the scovector $V = [\texttt{+++x+}]$. Using now the intersection criterion of Property 12.4, one can observe that two `Gas` homodromies, $\text{rep}(\Omega_5) = [\texttt{+xx-+}]$ and $\text{rep}(\Omega_9) = [\texttt{+++0+}]$, and the `SuperFluid` homodromy, $\text{rep}(\Omega_4) = [\texttt{+++++}]$, combinatorially intersect the scovector V. *We conclude that when the temperature is very high and the pressure is unknown, the only compatible results will be the* `Gas` *and* `SuperFluid` phases.

12.4.3 *Boundary Test Data Generation*

Boundary test data generation is the quantitative side of the model. Its principle resides in the transformation of combinatorial operators $(\vee, \wedge)$ in real valued operations. Cocircuits constitute the combinatorial and numerical buildings blocks of the model. Rephrasing it, in software testing teminology: by composing a restricted set of test points, it is possible to generate, automatically, new limit test points on the borders of all the possible execution paths of the algorithm. Test data generation takes two steps.

- Step 1: Combinatorial generation of a specific region B.
- Step 2: Numerical generation of a point inside B, using rational valued operations.

For this sake, we extensively use the border notion of Sec. 12.3.6. Let us consider a scovector U and its border $\partial(U)$. To each cocircuit or scovector C in $\partial(U)$, can be arbitrarily associated, a real vector, named sample(C), realizing C in region(C). Such a real vector can be easily computed from the initial equations (if C is a cocircuit, then sample(C) can be any real vector spanning the half-line contained in the line $\bigcap_{e\in E, C(e)=0} H_e$, in the same side as the region $\bigcap_{e\in E, C(e)\neq 0} H_e^{C(e)}$). For instance, samples of the cocircuits of our example (Figs. 12.2 and 12.6) are given in Table 12.2.

Then, by Theorem 12.2, a real vector in region(U) realizing $U = \bigvee_{C\in\partial(U)} C$ can be any combination of the real vectors sample(C) for $C \in \partial(U)$ with positive coefficients: sample(U) $= \sum_{C\in\partial(U)} \alpha_C$.sample($C$), with $\alpha_C \in \mathbb{R}^{*+}$ for all $C \in \partial(U)$. Furthermore, real vectors in the region in

Table 12.2 The 7 basic cocircuits of the toy model and the coordinates of their associated real samples. Cocircuit γ_1 = [00--+] means that, at the point $\gamma_1(0, 0, 1)$, e_1 and e_2 are zero, and e_3 and e_4 are negative. The fifth sign expresses its position at infinity (the positive sign means that the point is inside the specifications). The sample coordinates are given in the 3-dimensional vector space containing the 2-dimensional affine input space. The third coordinate equals 1 for points inside the specifications and 0 for the points "at infinity" (see General setting in Sec. 12.3.1). The 7 samples are represented in Figs. 12.2 and 12.11.

cocircuit	`coordinates`	`cocircuit`	`coordinates`
γ_1 `[00--+]`	`[0, 0, 1]`	γ_2 `[0+00+]`	`[600, 30, 1]`
c_1 `[0---0]`	`[-1000, -50, 0]`	c_2 `[-0-+0]`	`[-100, 1000, 0]`
c_3 `[+++00]`	`[0.1, 0, 0]`	c_4 `[0+++0]`	`[2, 0.1, 0]`
c_5 `[-+0+0]`	`[1, 0.1, 0]`		

the neighborhood of these boundary vectors can be generated (for instance) as sample$(C) + \varepsilon$.sample(U) for $C \in \partial(U)$ and for a small $\varepsilon \in \mathbb{R}^{*+}$. (In the case of an initial affine space, one has to divide all coordinates by the coordinate corresponding to the element at infinity, in order to get points inside the specifications.)

In this way, one can compute real vectors at all boundaries between homodromies as well as at the neighborhood of all boundaries in all homodromies that they bound. Such a set of real vectors, serving as a relevant limit test data, is illustrated with grey dots in Fig. 12.11, obtained from the fragmentation given in Fig. 12.6 for the toy example. Real coordinates of test data obtained in the above way are given in Table 12.1.

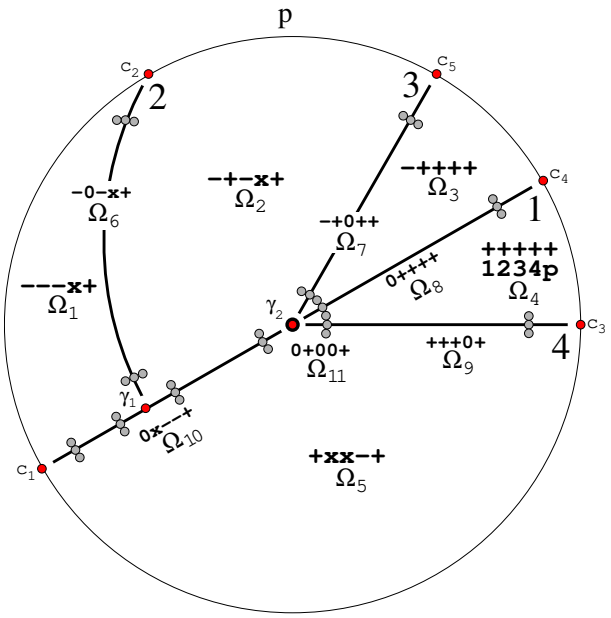


Fig. 12.11 Using scovectors and their borders to generate real vectors in homodromies of a fragmentation, in their boundaries, and in the neighborhood of their boundaries (Sec. 12.3.6), for the toy example of Sec. 12.2 and Fig. 12.2. The homodromies $\Omega_1, \ldots, \Omega_{11}$ correspond to the fragmentation of Fig. 12.6 and Table 12.1. The border cocircuits are represented by points denoted γ_1, γ_2, c_1, c_2, c_3, c_4, c_5, consistently with Fig. 12.2 and Table 12.2. Grey dots represent samples (or test points) in homodromies, their boundaries, and their neighborhood; they are obtained by barycentric combinations of cocircuit points.

We now present some applications of the previous general construction to the computation of limit test points. The qualitative execution, in the previous section, permitted us to conclude that, *when temperature is very close to zero and pressure has a normal positive value, algorithm yields the* `Liquid` phase. Suppose now that this result contradicts the specifications which predict that, in that case, one should obtain the 'ice' result (the `Solid` phase). In other words, the analyzed algorithm contains a defect (bug).

The tester thus decides to examine what happens at temperatures close to zero in the boundary of the `Solid` homodromy Ω_1 and the `Liquid` homodromy Ω_2. In other words, she wishes to obtain boundary inputs that lye exactly on the Solidification/Liquefaction surface.

- Step 1: Starting from Table 12.1, $\Omega_1 = $ `[--xx+]` and $\Omega_2 = $ `[-+-x+]`, one can compute their borders ($\partial(\Omega_1) = \{c_1, \gamma_1, c_2\}$ and $\partial(\Omega_2) = \{\gamma_1, \gamma_2, c_4, c_2\}$) hence, their representatives, getting $\text{rep}(\Omega_1) = $ `[---x+]` and $\text{rep}(\Omega_2) = $ `[-+-x+]`. Using Property 12.4, one can calculate their common face with the meet $\wedge$ operator: $\text{rep}(\Omega_1) \wedge \text{rep}(\Omega_2) = $ `[---x+]` $\wedge$ `[-+-x+]` $= $ `[-0-x+]` $= \text{rep}(\Omega_6)$. In other words, the homodromy Ω_6 (the open segment $]\gamma_1, c_2[$), forms the solidification/liquefaction separating surface between the two phases.
- Step 2: Since the tester wishes to examine what happens with a small temperature and a positive pressure, she may ask for a test data that is close to $\gamma_1(0,0)$, still staying inside Ω_6. Taking as coefficients (for the convex hull combination) two strictly positive rationals λ_1 and λ_2 with $\lambda_1 + \lambda_2 = 1$, with, say, $\lambda_1 = 9999 \times 10^{-4}$, "much closer" to 1, than $\lambda_2 = 10^{-4}$. The join $\vee$ operation is now translated into a convex sum of rational coordinates. We obtain a boundary test data, $T_1 = \lambda_1 \cdot \text{sample}(\gamma_1) + \lambda_2 \cdot \text{sample}(c_2) = 9999 \cdot 10^{-4}$`[0,0,1]`$+10^{-4}$`[-100,1000,0]` $=$ `[-0.01,0.1,0.9999]`. Transforming the last coordinates in affine coordinates (dividing by the non-zero infinite third coordinate) one gets the test data $T_1 = $ `[-0.010001, 0.10001]`. One can check that T_1 nullifies the equation e_2, and still lies inside the `Solid` homodromy Ω_6. Exchanging λ_1 with λ_2 would still yield a point of Ω_6 but close to the specification frame.

What happens if one wishes to have an equivalent point of T_1 (close to γ_1) but in the `Liquid` side? Like previously:

- Step 1: The `Liquid` homodromy Ω_2 = [`-+-x+`] admits 4 border cocircuits, $\{\gamma_1, \gamma_2, c_4, c_2\}$, with $\text{rep}(\Omega_2)$ = [`-+x+`] = $\gamma_1 \vee \gamma_2 \vee c_4 \vee c_2$.
- Step 2: For producing a limit test data point, T_2 close to $\gamma_1(0,0)$, but staying inside the `Liquid` homodromy we choose four positive rationals $\lambda_{1..4}$ such that their sum equals 1. Choosing say $\lambda_1 = 997 \cdot 10^{-5}$ and $\lambda_2 = \lambda_3 = \lambda_4 = 10^{-5}$, we obtain: $T_2 = \lambda_1 \cdot \text{sample}(\gamma_1) + \lambda_2 \cdot \text{sample}(\gamma_2) + \lambda_3 \cdot \text{sample}(c_4) + \lambda_4 \cdot \text{sample}(c_2)$ = $997 \cdot 10^{-5}$`[0,0,1]` + 10^{-5}`[600,30,1]` + 10^{-5}`[2,0.1,0]` + 10^{-5}`[-100,1000,0]` = `[0.00502, 0.001301, 0.00998]`. In affine coordinates one gets the test data T_2 = `[0.503006, 0.130361]`, yielding a `Liquid` output.

12.5 Conclusion

We presented an algorithmic qualitative model intended to assist the tester during the unit testing process. This model permits the visualization, in an abstract level, of the algorithmic behaviour. The spatial properties are based on oriented matroids. The propagation of orders-of-magnitude enables the validation of abstract properties concerning boundary behaviour. The combinatorial properties of cocircuits are finally used to generate concrete numeric test sets on any critical surface of any dimension.

Our approach is a technique allowing the automatic generation of test data. As we stressed in the introduction, this constitutes a critical open issue in the software engineering process, thus many other techniques have been proposed to tackle this problem. Note however that our testing data generation approach is not based on a structural coverage objective (i.e. cover all the statements of the source code), nor model based (i.e. use of an oriented graph), nor heuristic (like evolutionary techniques). It produces, formally and exhaustively (under certain assumptions and thanks to the mathematical theory presented in the previous sections), all the feasible execution paths of the source code. It is thus stronger (in the sense of [26]) than any other structural (path, branch, du-path, etc.) coverage objective.

This work is still at a prototype stage and is intended for software units whose conditions depend on decimal or rational inputs. Note however, that this restriction does not concern outputs which can be of any arbitrary type, as long as one disposes of an equivalence relation to aggregate them.

As a first experiment to evaluate its usefulness in the development process, the prototype was used in a first year Python programming course. The pedagogical objective was twofold: first, sensitize the students to the testing process by making them compare their (manually produced) test data with the automatic ones. Second, thanks to the envision graph, it gives to the students a more abstract and visual view of what was effectively coded, that can be compared with their qualitative understanding of the initial specifications.

A limitation obviously concerns the number of equations that the tool can analyze (limited in our prototype to 50), a number which can become very large in the case of large arrays and/or nested iterations. Note however, that the tool disposes of several filtering (slicing) options that can reduce drastically the number of studied paths.

We are currently working on several technical and theoretical future developments: the construction of a qualitative formal proof engine that checks properties by examining all possible paths (or in certain directions) of the homodromy graph; a qualitative simulation engine (that could be compared to a qualitative debugger) coupled with a decorated source code browser, which visualizes, step by step, the path followed on the homodromy graph according to qualitative input directions or magnitudes; and further useful mathematical properties in terms of oriented matroids.

Acknowledgments

This research was supported by: the OMSMO Project (Oriented Matroids for Shape Modeling) - Grant "Chercheur d'avenir 2015" (Région Occitanie & Fonds Européen de Développement Régional FEDER); the ANR Grant DISTANCIA (Metric Graph Theory, ANR-17-CE40-0015); and the ARCHIMEDES III (Greek minister of Research) project (support for TEI Larissa, No. 16, 2010: Application of Genetic algorithms and Qualitative Reasoning for intelligent software testing).

References

[1] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas and K. Karapoulios, Application of genetic algorithms to software testing (application des algorithmes génétiques au test des logiciels), in *5th International Conference on Software Engineering and its Applications, Toulouse, France*, pp. 625–636 (1992).

[2] M. Schoenauer and S. Xanthakis, Constrained GA optimization, in *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA*, pp. 573–580 (1993).
[3] R. P. Pargas, M. J. Harrold and R. R. Peck, Test data generation using genetic algorithms, *Software Testing, Verification and Reliability* **9**, pp. 263–282 (1999).
[4] P. McMinn, Search-based software test data generation: A survey, *Software Testing, Verification and Reliability* **14**, 2, pp. 105–156 (2004).
[5] O. Bühler and J. Wegener, Evolutionary functional testing, *Computers and Operations Research* **35**, pp. 3144–3160 (2008).
[6] S. Di Alesio, L. Briand, S. Nejati and A. Gotlieb, Combining genetic algorithms and constraint programming to support stress testing of task deadlines, *ACM Transactions on Software Engineering and Methodology* **25**, pp. 1–37 (2015).
[7] C. Sharma, S. Sabharwal and R. Sibal, A survey on software testing techniques using genetic algorithm, *International Journal of Computer Science Issues* **10** (2013).
[8] Z. Zhu and L. Jiao, Improving search-based software testing by constraint-based genetic operators, in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO 19. Association for Computing Machinery, New York, NY, USA, pp. 1435–1442 (2019).
[9] S. Xanthakis, S. Karapoulios, R. Pajot and A. Rozz, Immune system and fault tolerant computing, in *Artificial Evolution*, Vol. 1063. Lecture Notes in Computer Science, Springer-Verlag, pp. 181–197 (1996).
[10] Y. Jia and M. Harman, Constructing subtle faults using higher order mutation testing, in *8th International Working Conference on Source Code Analysis and Manipulation (SCAM 2008). Beijing, China, IEEE Computer Society* (2008).
[11] L. C. Briand, J. Feng and Y. Labiche, Using genetic algorithms and coupling measures to devise optimal integration test orders, in *14th IEEE Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy*, pp. 43–50 (2002).
[12] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer and R. S. Roos, Time aware test suite prioritization, in *International Symposium on Software Testing and Analysis (ISSTA 06). Portland, Maine, USA, ACM Press*, pp. 1–12 (2006).
[13] S. Yoo and M. Harman, Pareto efficient multi-objective test case selection, in *International Symposium on Software Testing and Analysis (ISSTA 07), ACM Press*, pp. 140–150 (2007).
[14] D. S. Weld and J. de Kleer, *Readings in Qualitative Reasoning about Physical Systems*. Morgan-Kaufman (1990).
[15] D. S. Weld and J. de Kleer, in D. S. Weld and J. de Kleer (eds.), *Qualitative Reasoning*. Series in Artificial Intelligence (1989).
[16] L. Travé-Massuyès and N. Piera, The orders of magnitude models as qualitative algebras, in *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence - Volume 2*, pp. 1261–1266 (1989).

[17] D. S. Weld, Exaggeration, in D. S. Weld and J. de Kleer (eds.), *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, pp. 417–421 (1990).
[18] S. Parsons and M. Dohnal, The qualitative and semiqualitative analysis of environmental problems, *Environmental Software* **10**, pp. 75–85 (1995).
[19] R. Moratz, *Qualitative Spatial Reasoning*. Encyclopedia of GIS, Editors: Shashi Shekhar, Hui Xiong, Xun Zhou (2017).
[20] K. D. Forbus, *Qualitative Representations How People Reason and Learn about the Continuous World*. MIT Press (2019).
[21] A. Björner, M. Las Vergnas, B. Sturmfels, N. White and G. Ziegler, *Oriented Matroids*, *Encyclopedia of Mathematics and Its Applications*, Vol. 46, 2nd edn. Cambridge University Press (1999).
[22] S. Xanthakis and E. Gioan, A qualitative reasoning model for software testing, based on oriented matroid theory, Full length journal paper (detailing and completing the present chapter) (In preparation).
[23] J. Edmonds and A. Mandel, *Topology of oriented matroids*. Ph.D. Thesis of A. Mandel, University of Waterloo (1982).
[24] G. Reeb, Analyse non standard (essai de vulgarisation), *Bulletin de l'APMEP* **32** (1981).
[25] R. Goldblatt, *Lectures on the Hyperreals. An introduction to nonstandard analysis*, Vol. 188. Graduate Texts in Mathematics. Springer-Verlag MR164 (1998).
[26] E. J. Weyuker, Comparing the effectiveness of testing techniques, in *Formal Methods and Testing*, Vol. 4949. Lecture Notes in Computer Science, Springer, pp. 271–291 (2008).