

David Ilsen

Algebraic and Combinatorial Algorithms for Translinear Network Synthesis

Vom Fachbereich Mathematik der Universität Kaiserslautern

zur Verleihung des akademischen Grades

Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.)

genehmigte Dissertation

1. Gutachter: Prof. Dr. G.-M. Greuel

2. Gutachter: Dr. W. A. Serdijn

Vollzug der Promotion: 19. Mai 2006

D 386

Contents

1. Introduction	4
2. Prerequisites	7
2.1. Notions from Graph Theory	7
2.2. The Toric Ideal of a Digraph	10
3. Translinear Network Theory	13
3.1. The Translinear Principle	13
3.1.1. The Static Translinear Principle	14
3.1.2. Motivation for a Catalog of Topologies	17
3.1.3. The Dynamic Translinear Principle	18
3.2. Translinear Decomposition of Polynomials	20
3.3. The Topology of Translinear Networks	22
3.3.1. Translinear Digraphs	22
3.3.2. Connection of Collectors	26
3.3.3. Insertion of Capacitances	28
3.4. The Interface of a Translinear Network	28
3.4.1. Outputs	28
3.4.2. Inputs and Ground Node Selection	29
3.5. Topologies for 4-terminal MOS transistors	30
4. Combinatorial Generation of Translinear Networks	32
4.1. Orderly Generation	32
4.1.1. Cataloging Problems	32
4.1.2. Canonicity and Augmentation	35
4.1.3. Generation of Canonical Words	37
4.2. Generation of Digraphs	40
4.3. Generation of Translinear Digraphs	44
4.3.1. Adjacency matrices of layer digraphs	44
4.3.2. Canonicity	45
4.3.3. Augmentation	46
4.3.4. Specialization for Translinear Digraphs	47
4.4. Generation of Collector Assignments	48
4.5. Cataloging Formal Networks	49

5. A Catalog of Topologies as a Synthesis Tool	52
5.1. Translinear Network Equations	52
5.1.1. Translinear Loop Equations	53
5.1.2. Node Equations	53
5.1.3. The Network Ideal	53
5.1.4. Elimination of Collector Currents	54
5.2. The Input Matrix	55
5.3. Solution of the Matching Problem	57
5.4. Final Network Check	58
5.4.1. Positivity Check of Collector Currents	58
5.4.2. Output Function Check	59
6. Example Application	60
7. Conclusion	64
A. The Static Formal Translinear Networks with up to 6 Transistors	66
B. Implementations	69
B.1. Overview of the Implementations	69
B.2. Combinatorial Generation of Formal Networks	71
B.2.1. The class “matrix”	71
B.2.2. The classes “permutation” and “multiperm”	72
B.2.3. The class “mpsims”	73
B.2.4. The class “ranked”	74
B.2.5. The class “network”	75
B.3. Sample <code>tlgen</code> Output	77
B.4. Match Finding	80
B.5. Final Network Checks	82
Bibliography	83

1. Introduction

This thesis contains the mathematical treatment of a special class of analog microelectronic circuits called *translinear circuits*. The goal is to provide foundations of a new coherent synthesis approach for this class of circuits. The mathematical methods of the suggested synthesis approach come from graph theory, combinatorics, and from algebraic geometry, in particular symbolic methods from computer algebra.

Translinear circuits [Gil75, Gil96] form a very special class of analog circuits, because they rely on nonlinear device models, but still allow a very structured approach to network¹ analysis and synthesis. Thus, translinear circuits play the role of a bridge between the “unknown space” of nonlinear circuit theory and the very well exploited domain of linear circuit theory.

The nonlinear equations describing the behavior of translinear circuits possess a strong algebraic structure that is nonetheless flexible enough for a wide range of nonlinear functionality. Furthermore, translinear circuits offer several technical advantages like high functional density, low supply voltage and insensitivity to temperature.

This unique profile is the reason that several authors consider translinear networks as the key to systematic synthesis methods for nonlinear circuits [DRVRV99, RVDRHV95, Ser05].²

This thesis proposes the usage of a computer-generated catalog of translinear network topologies as a synthesis tool. The idea to compile such a catalog has grown from the observation that on the one hand, the topology of a translinear network must satisfy strong constraints which severely limit the number of “admissible” topologies, in particular for networks with few transistors, and on the other hand, the topology of a translinear network already fixes its essential behavior, at least for static networks, because the so-called *translinear principle* requires the continuous parameters of all transistors to be the same.

Even though the admissible topologies are heavily restricted, it is of course a highly nontrivial task to compile such a catalog. Combinatorial techniques have been adapted to undertake this task.

The idea to utilize synthetic lists of network topologies is not new in analog circuit design: Catalogs of VCCS topologies are used for CMOS circuit design by E. Klumperink and others [Klu97, KBN01, Sch02, Sch04].

¹In this thesis, a “circuit” means electronics hardware, whereas a “network” means its mathematical model.

²The “Wiley Encyclopedia of Electrical and Electronics Engineering” expresses the prominent role of translinear circuits by the fact that the entire entry for “nonlinear circuits” consists only of a reference to “translinear circuits”.

1. Introduction

In a catalog of translinear network topologies, prototype network equations can be stored along with each topology. When a circuit with a specified behavior is to be designed, one can search the catalog for a network whose equations can be matched with the desired behavior.

In this context, two algebraic problems arise: To set up a meaningful equation for a network in the catalog, an elimination of variables must be performed, and to test whether a prototype equation from the catalog and a specified equation of desired behavior can be “matched”, a complex system of polynomial equations must be solved, where the solutions are restricted to a finite set of integers. Sophisticated algorithms from computer algebra are applied in both cases to perform the symbolic computations.

All mentioned algorithmic methods have been implemented and successfully applied to actual design problems at Analog Microelectronics GmbH (in the following: AMG), Mainz.

The thesis is organized as follows:

Chapter 2 collects some graph-theoretic and algebraic background that will be needed in the other chapters.

Chapter 3 first reviews the basic concepts and facts about translinear circuits, then gives an analysis of their topology and develops some abstract notions to model the topology in terms of graph theory.

Chapter 4 is about techniques for producing catalogs of combinatorial objects and the specialization of these techniques to list translinear network topologies exhaustively for a given number of transistors.

The concerns of Chapter 5 are the structure of the equations describing a translinear network’s behavior, and the algebraic problems which have to be solved when the network catalog is to be equipped with prototype network equations and when it is searched for a network with a particular behavior.

Chapter 6 reports about the successful application of the developed synthesis methodology in the design of a new humidity sensor system of AMG.

As an impression of the catalog of networks that is produced, an overview over the static formal translinear networks with 6 or less transistors is given in Appendix A.

The algorithms presented in this thesis for building and searching a catalog of translinear network topologies have been implemented using C++, SINGULAR, and MATHEMATICA. Some details and comments of the implementations are included in Appendix B.

Acknowledgements

My first thanks go to my advisor Prof. Dr. Gert-Martin Greuel. His advice was of excellent scientific and personal quality.

The collaboration with Analog Microelectronics GmbH was essential for my research. I am very much indebted to development engineer Ernst Josef Roebbers for the initiation of the research topic, and for plenty of discussions about my work and its applications. Thanks are also due to managing director Dr. N. Rauch for supplementary explanations and for making the collaboration possible.

I thank Dr. Wouter Serdijn for his helpful comments, in particular for inspiring the topological considerations for MOS translinear circuits.

Further, I thank Prof. Dr. Gerhard Pfister for several discussions, and the AnalogInsydes team at Fraunhofer-ITWM for the frequent meetings at which they shared their circuit analysis experience with me.

I am grateful for the honor to receive a scholarship of the DFG-Graduiertenkolleg “Mathematik und Praxis” in Kaiserslautern, which provided the financial support for my research.

For last-minute proofreading, I thank Britta Späth, Alexander Dreyer, Hannah Markwig, Burkhard Ilse, and Dr. K. Mann.

Music kept my spirits alive when math was too frustrating. Thanks go to my friends from “Haste Töne”.

Without the moral support from Kerstin and from all of my parents, I wouldn't have been able to complete this thesis. Thanks.

I dedicate my work to the remembrance of Magdalene Ilse, my wonderful “fabulous grandmother”, who died on August 28th, 2005. She has a great part in my personal development, not least by her accompaniment of my first steps as a mathematician in Dortmund.

Muja, ich danke Dir für all die Kraft und Wärme und werde sie immer in meinem Herzen behalten!

2. Prerequisites

For the algebraic notions used in this thesis, we refer to textbooks, e.g. [CLO97, FS83, Lan94, GP02]. For some of the notions we need from Graph Theory, the literature shows subtle inconsistencies, so we clarify these notions in Section 2.1. Section 2.2 collects some nonstandard constructions and facts about toric ideals, which will be needed in Chapter 5.

2.1. Notions from Graph Theory

Definition 2.1. A **directed graph** or **digraph** is a triple $G = (V, E, \iota)$ of two sets V and E and a map $\iota : E \rightarrow V \times V$. The elements of V are called **nodes** or **vertices**, the elements of E are called **branches**. ι is called the **incidence map** of G . For a branch $e \in E$ with $\iota(e) = (v_1, v_2)$, v_1 is called its **tail node** or **start vertex** and also denoted by $\text{tail}(e)$, and v_2 is called the **head node** or **terminal vertex** of e and also denoted by $\text{head}(e)$. We say that a branch e “points from $\text{tail}(e)$ to $\text{head}(e)$ ”.

Remark 2.2. Definition 2.1 allows **parallel branches** and **self-loops** in a digraph. (A pair of parallel branches consists of distinct elements $e, e' \in E$ with $\iota(e) = \iota(e')$; a self-loop is a branch $e \in E$ with $\text{tail}(e) = \text{head}(e)$.) Authors in graph theory frequently exclude both in digraphs. If parallel branches or self-loops occur, they rather speak of a *directed multigraph*. Here, we deliberately chose Definition 2.1 to be as it is.

Definition 2.3. A digraph $G = (V, E, \iota)$ is called **finite** if both V and E are finite sets.

Remark 2.4. In all examples of this thesis, digraphs are finite.

If G is a directed graph, its set of nodes will also be denoted by $V(G)$, and its set of branches will also be denoted by $E(G)$.

Definition 2.5. A **walk** in a digraph G is an alternating finite sequence

$$W = (v_0, e_1, v_1, \dots, e_{l-1}, v_{l-1}, e_l, v_l)$$

of nodes and branches such that for each $j = 1, \dots, l$, either $\iota(e_j) = (v_{j-1}, v_j)$ or $\iota(e_j) = (v_j, v_{j-1})$. In the former case, e_j is called a **forward branch**, in the latter case it is called a **backward branch** of W . We say that W is a walk **from** v_0 **to** v_l . The number $l \in \mathbb{N}_0$ is called the **length** of W . If the nodes v_0, \dots, v_l (and thus also the branches)

2. Prerequisites

are pairwise distinct, W is called a **path**. If $v_0 = v_l$, W is called a **cycle**. A cycle $(v_0, e_1, \dots, v_{l-1}, e_l, v_0)$ with pairwise distinct nodes v_0, \dots, v_{l-1} is called a **loop**.

Sometimes we use the notation $E(W) := \{e_1, \dots, e_l\}$ for the set of branches appearing in a walk $W = (v_0, e_1, v_1, \dots, e_l, v_l)$. The walk $\overleftarrow{W} := (v_l, e_l, v_{l-1}, e_{l-1}, \dots, v_1, e_1, v_0)$ is the **reversed walk** (path/cycle/loop, resp.) of W .

If a walk (path/cycle/loop) W has no backward branches, we call it a **directed walk** (**path/cycle/loop**).

Definition 2.6. Let G be a digraph, let $W = (v_0, e_1, v_1, \dots, e_l, v_l)$ be a walk in G , and let $e \in E(G)$ be any branch of G . For $j = 1, \dots, l$, define

$$\mu(W, e, j) := \begin{cases} 1 & \text{if } e = e_j \text{ and } \iota(e) = (v_{j-1}, v_j), \\ -1 & \text{if } e = e_j \text{ and } \iota(e) = (v_j, v_{j-1}), \\ 0 & \text{if } e \neq e_j. \end{cases}$$

The **incidence index of e in W** is

$$\mu(W, e) := \sum_{j=1}^l \mu(W, e, j).$$

Remark 2.7. In effect, $\mu(W, e)$ is the number of times e appears as a forward branch in W minus the number of times e appears as a backward branch in W . If W is a path or a loop, $\mu(W, e)$ is either 1, -1 , or 0. Note that $\mu(\overleftarrow{W}, e) = -\mu(W, e)$ for every walk W and every branch e .

Definition 2.8. Let $W = (v_0, e_1, v_1, \dots, e_l, v_l)$ and $W' = (v'_0, e'_1, v'_1, \dots, e'_{l'}, v'_{l'})$ be two walks such that $v_l = v'_0$. Then we define the walk

$$W \star W' := (v_0, e_1, v_1, \dots, e_l, v_l, e'_1, v'_1, \dots, e'_{l'}, v'_{l'})$$

from v_0 to $v'_{l'}$. Furthermore, for every branch e with $\text{tail}(e) = v_l$, we define the walk

$$W \star e := (v_0, e_1, v_1, \dots, e_l, v_l, e, \text{head}(e))$$

from v_0 to $\text{head}(e)$.

Definition 2.9. We call a digraph G **connected**, if for any two nodes $v, v' \in V(G)$, there is a walk in G from v to v' .

Definition 2.10. Let $G = (V, E, \iota)$ be a digraph, and let $\bar{V} \subseteq V$ be a subset of its nodes. We define the digraph $G|\bar{V} := (\bar{V}, \bar{E}, \bar{\iota})$ by $\bar{E} := \{e \in E \mid \iota(e) \in \bar{V} \times \bar{V}\}$ and $\bar{\iota}(e) := \iota(e)$ for $e \in \bar{E}$. If $G|\bar{V}$ is a connected digraph, and if furthermore there is no other node subset V' such that $\bar{V} \subsetneq V' \subseteq V$ and $G|V'$ is connected, then $G|\bar{V}$ is called a **connected component of G** .

2. Prerequisites

Lemma 2.11. *For every finite digraph G , there is a finite number of subsets $V_1, \dots, V_s \subset V(G)$ such that*

1. *The connected components of G are exactly $G|V_1, \dots, G|V_s$.*
2. *$V_1 \cup \dots \cup V_s = V(G)$.*
3. *$V_j \cap V_k = \emptyset$ for $j, k = 1, \dots, s$ and $j \neq k$.*

Proof. The existence of one connected component $G|V_1$ is easily shown. Continue with $G|(V \setminus V_1)$ to obtain V_2 , and so on. \square

Definition 2.12. G is called **biconnected** if it is connected and remains so after the removal of an arbitrary node, that is, if for every $v \in V$, the digraph $G|(V \setminus \{v\})$ is connected.

Definition 2.13. Let G be a directed graph and let $V(G) = \{v_1, \dots, v_n\}$. The **adjacency matrix** $A^G \in \mathbb{N}_0^{n \times n}$ of G is defined as follows: For $j, k = 1, \dots, n$, the entry A_{jk}^G is the number of different branches $e \in E(G)$ with $\iota(e) = (v_j, v_k)$.

Remark 2.14. Obviously, the adjacency matrix depends on the order which the nodes v_1, \dots, v_n are indexed with. Because in most cases it is clear by notation how the nodes are ordered, it is common practice in textbooks to neglect this dependence. We follow this practice in most parts of this thesis. However, we emphasize here the non-uniqueness of the adjacency matrix, because it will be an important issue in Chapter 4.

Definition 2.15. Let G be a directed graph, let $V(G) = \{v_1, \dots, v_n\}$ and let $E(G) = \{e_1, \dots, e_b\}$. The **incidence matrix** $M^G \in \mathbb{Z}^{b \times n}$ of G is defined as follows: For $j = 1, \dots, b$ and $k = 1, \dots, n$,

$$M_{jk}^G := \begin{cases} 0, & \text{if } \text{head}(e_j) \neq v_k \neq \text{tail}(e_j) \text{ or } \text{head}(e_j) = \text{tail}(e_j) = v_k, \\ 1, & \text{if } \text{head}(e_j) = v_k \neq \text{tail}(e_j), \\ -1, & \text{if } \text{head}(e_j) \neq v_k = \text{tail}(e_j). \end{cases}$$

Remark 2.16. The incidence matrix depends on the ordering of the nodes as well as on the ordering of the branches.

Definition 2.17. Let G be a directed graph with $E(G) = \{e_1, \dots, e_b\}$, and let S be a loop of G . The **loop incidence vector** of S is

$$u_S := \begin{pmatrix} \mu(S, e_1) \\ \vdots \\ \mu(S, e_b) \end{pmatrix}.$$

Notation 2.18. We denote the transposed matrix of a matrix A by A^t .

Lemma 2.19. *For every loop S of a digraph G , $u_S \in \ker (M^G)^t$.*

2. Prerequisites

Proof. Straightforward. □

Proposition 2.20. *For every $u = (u_1, \dots, u_b)^t \in \ker(M^G)^t \subset \mathbb{Z}^b$, there are loops S_1, \dots, S_r , $r \in \mathbb{N}_0$ such that $u = u_{S_1} + \dots + u_{S_r}$ and for every $l = 1, \dots, r$:*

1. *Whenever $u_j = 0$, then $\mu(S_l, e_j) = 0$.*
2. *Whenever $u_j > 0$, then $\mu(S_l, e_j) \geq 0$.*
3. *Whenever $u_j < 0$, then $\mu(S_l, e_j) \leq 0$.*

Proof. (Sketch.) By induction on $|u| := \sum_{j=1}^b |u_j|$. Carefully selecting one branch after the other, we can construct a loop S_1 such that 1.-3. hold (for $l = 1$). Then we continue with $u - u_{S_1}$ instead of u . □

Corollary 2.21. *$\ker(M^G)^t$ is generated by the loop incidence vectors.*

Definition 2.22. A tuple of loops S_1, \dots, S_r is called a **system of fundamental loops** if u_{S_1}, \dots, u_{S_r} form a basis of $\ker(M^G)$.

2.2. The Toric Ideal of a Digraph

To any directed graph G , we can associate its *toric ideal*, an algebraic object that carries the essential information about the loop structure of G . (In this way, the toric ideal of G is similar, and in fact closely related to the *cycle space* and to the *fundamental groups* of G .) In the context of this thesis, the toric ideal of a digraph is of particular interest, because in the case of a translinear digraph (see Definition 3.5) it actually consists of the translinear loop equations of any network based on that digraph. We will come back to the role of this toric ideal of a translinear network in Subsection 5.1.1.

Of course, there is a general notion of a toric ideal, independent of digraphs and networks. Toric ideals are not only the algebraic building blocks giving rise to the very rich *toric geometry* (see [Ful97] for an introduction), they also have applications in integer programming and combinatorics and thus attract much attention from the computational viewpoint [Stu97, The99].

Definition 2.23. Let k be any field and let $M = (m_{ij}) \in \mathbb{Z}^{n \times b}$ be an integer matrix. The **toric ideal of M over k** , denoted by I_A , is the kernel of the k -algebra-homomorphism

$$\begin{aligned} k[x_1, \dots, x_b] &\rightarrow k[t_1, \dots, t_n, t_1^{-1}, \dots, t_n^{-1}], \\ x_j &\mapsto t_1^{m_{1j}} \dots t_n^{m_{nj}}. \end{aligned}$$

Remark 2.24. Any toric ideal is prime, since it is the kernel of a homomorphism into an integral domain.

2. Prerequisites

Notation 2.25. For $u = (u_1, \dots, u_b) \in \mathbb{Z}^b$, define the monomials $m_u^+ := \prod_{u_j > 0} x_j^{u_j}$, $m_u^- := \prod_{u_j < 0} x_j^{-u_j}$, and the binomial $B_u := m_u^+ - m_u^- \in k[x_1, \dots, x_b]$.

Lemma 2.26. I_A is generated by $\{B_u \mid u \in \ker(M) \subset \mathbb{Z}^b\}$.

Proof. [Stu97, Corollary 4.3] □

Notation 2.27. For an ideal $J \subset k[x_1, \dots, x_b]$, we denote its saturation with respect to all variables by $\text{sat}(J)$. That is, $\text{sat}(J) = \{f \mid \exists \text{ monomial } m : mf \in J\}$.

Lemma 2.28. If u_1, \dots, u_s form a basis of $\ker(M)$, then $I_A = \text{sat}(\langle B_{u_1}, \dots, B_{u_s} \rangle)$.

Proof. [Stu97, Lemma 12.2] □

We now focus our attention on the toric ideal of a digraph.

Several authors have examined the toric ideal of a general *undirected* graph [SVV94, dLST95, OH99]. Ishizeki and Imai have considered toric ideals of acyclic digraphs and Gröbner bases of them [Ish00b, II00b], their publications seem to be the only ones where the toric ideal of a *digraph* has been mentioned hitherto.

Definition 2.29. Let G be a directed graph. The **toric ideal of G** , denoted by I_G , is the toric ideal of its transposed incidence matrix $(M^G)^t$ over \mathbb{Q} .

Remark 2.30. If we identify $\mathbb{Q}[x_1, \dots, x_b]$ with the free \mathbb{Q} -algebra on the set of branches, and $\mathbb{Q}[t_1, \dots, t_n, t_1^{-1}, \dots, t_n^{-1}]$ with the free \mathbb{Q} -algebra on the set of nodes and their inverses, then I_G is the kernel of the homomorphism defined by $e \mapsto \text{head}(e)(\text{tail}(e))^{-1}$ for each branch e .

Notation 2.31. For the incidence vector u_S of a loop S , we abbreviate $m_S^+ := m_{u_S}^+$, $m_S^- := m_{u_S}^-$, and $B_S := B_{u_S}$.

Lemma 2.32. For a digraph G ,

$$I_G = \langle B_S \mid S \text{ loop of } G \rangle$$

Proof. In view of Lemma 2.26, it suffices to show that $B_u \in \langle B_S \mid S \text{ loop of } G \rangle$ for every $u \in \ker(M^G)^t$. From Proposition 2.20 we obtain loops S_1, \dots, S_r such that $u = \sum_{i=1}^r u_{S_i}$, and $m_{S_i}^+$ and $m_{S_j}^-$ are coprime for all $i, j = 1, \dots, r$. Thus $m_u^+ = \prod_{i=1}^r m_{S_i}^+$ and $m_u^- = \prod_{i=1}^r m_{S_i}^-$. We can write B_u as

$$B_u = \sum_{i=1}^r \left(\prod_{j < i} m_{S_j}^- \right) \left(\prod_{j > i} m_{S_j}^+ \right) B_{S_i}.$$

□

2. Prerequisites

The last lemma of this chapter provides a possibility to determine a finite generating set for the toric ideal of a digraph:

Lemma 2.33. *Let u_1, \dots, u_s be the loop incidence vectors of a system of fundamental loops of G . Then $I_G = \text{sat}(\langle B_{u_1}, \dots, B_{u_s} \rangle)$.*

Proof. Follows from Lemma 2.28. □

3. Translinear Network Theory

As mentioned earlier, the goal of this thesis is a coherent and well-structured synthesis methodology for translinear circuits, based on a catalog of topologies.

To give proper foundations for the new synthesis approach, we develop in this chapter (after an introduction to translinear circuits and an earlier synthesis approach in Sections 3.1 and 3.2) a mathematically rigorous perception of translinear circuits. In particular, we give a clean definition of a “translinear network” from a topological point of view.

3.1. The Translinear Principle

This section gives a review of the so-called translinear principle, the functional principle of translinear circuits. It has been formulated and given its name by Barrie Gilbert in 1975 [Gil75].

The translinear principle relies on an exponential voltage-to-current relationship of certain devices. The original “translinear device” is the bipolar NPN transistor, other devices with valid exponential models are diodes, PNP transistors and MOS transistors operating in weak inversion [Wie93]. Recently, an emulation of a bipolar transistor has been proposed [DBS04], where a subnetwork structure of three CMOS transistors and one diode shows the necessary exponential behavior.

In our circuit diagrams, we will use the symbol of a bipolar NPN transistor, shown in Figure 3.1, to represent an abstract “translinear device”, and we will simply use the term “transistor” for such an abstract device. It follows from the above that several different silicon implementations of a “transistor” are possible.

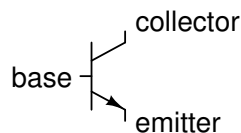


Figure 3.1.: The symbol for a bipolar NPN transistor, our placeholder for a “translinear device”.

3. Translinear Network Theory

The ideal exponential model of a transistor is given by the equation

$$I_{CE} = I_S e^{V_{BE}/U_T}, \quad (3.1)$$

saying that its collector current I_{CE} (the current from collector to emitter) is exponentially dependent on the base voltage V_{BE} (the voltage between base and emitter). In this model, I_S and U_T are device- and operation-dependent parameters called *saturation current* and *thermal voltage*, respectively. It is assumed that $I_S > 0$ and $U_T > 0$.

We usually make the additional model assumption that the base current (the current from base to emitter) of a transistor is zero.

3.1.1. The Static Translinear Principle

The key structures of translinear networks are so-called **translinear loops**. We call a loop W of the network digraph a translinear loop if it satisfies the following three properties:

- W consists exclusively of base-emitter branches of transistor.
- All transistors involved share the same pair (I_S, U_T) of parameters.
- W consists of as many forward branches as backward branches. Remember that we regard the branches to point “from base to emitter”.

Figure 3.2 shows two examples for a translinear loop.

The interesting property of translinear loops is that due to the exponential transistor model, we can deduce a multiplicative relation of collector currents from KIRCHHOFF’s Voltage Law (KVL): Denote the base voltages of the transistors in W -forward orientation by V_1^f, \dots, V_r^f , the base voltages of the transistors in W -backward orientation by V_1^b, \dots, V_r^b . Then KVL for W reads

$$V_1^f + \dots + V_r^f = V_1^b + \dots + V_r^b.$$

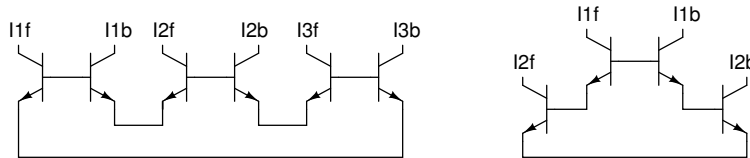


Figure 3.2.: Examples for translinear loops.

3. Translinear Network Theory

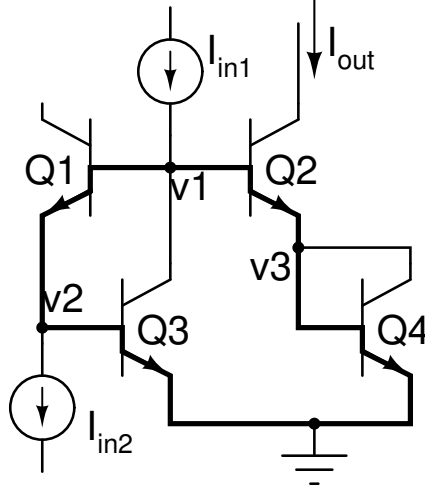


Figure 3.3.: A geometric mean circuit

Taking advantage of the common parameters, we deduce

$$e^{V_1^f/U_T} \cdot \dots \cdot e^{V_r^f/U_T} = e^{V_1^b/U_T} \cdot \dots \cdot e^{V_r^b/U_T}$$

and multiplication by I_S^r yields

$$\left(I_S e^{V_1^f/U_T}\right) \cdot \dots \cdot \left(I_S e^{V_r^f/U_T}\right) = \left(I_S e^{V_1^b/U_T}\right) \cdot \dots \cdot \left(I_S e^{V_r^b/U_T}\right).$$

Considering the model equation eqn. (3.1), this means exactly

$$I_1^f \cdot \dots \cdot I_r^f = I_1^b \cdot \dots \cdot I_r^b, \quad (3.2)$$

where I_1^f, \dots, I_r^f and I_1^b, \dots, I_r^b denote the collector currents of the transistors whose base-emitter branches are W -forward or W -backward, respectively.

Remark 3.1. Note that I_S and U_T don't occur any more in eqn. (3.2). This means that the relation between the collector currents holds independently of these parameters, provided they are indeed common. One nice effect of this is that translinear networks are essentially temperature-insensitive.

Example 3.2. The loop indicated by thick lines in Figure 3.3 is a translinear one, being made up of the base-emitter branches of transistors Q_1, \dots, Q_4 . (We assume that U_T and I_S coincide for the four transistors.) Application of the translinear principle yields¹

$$I_1 \cdot I_3 = I_2 \cdot I_4. \quad (3.3)$$

¹Here we simply denote the collector current of a transistor Q_j by I_j . We will stick to this convention in the following examples, too.

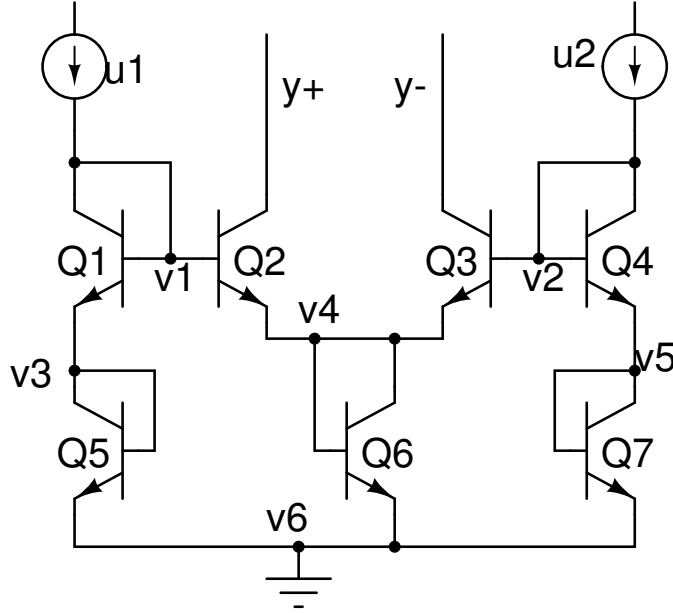


Figure 3.4.: A translinear frequency doubling network. (Since its first publication by GENIN and KONN in 1979 [GK79], this network has become a very prominent example application of the translinear principle.)

Now remember our assumption that base currents are zero. Taking this into account, Kirchhoff's Current Law (KCL) for v_1 means that $I_3 = I_{in1}$. Similarly for v_2 : $I_1 = I_{in2}$, and for v_3 : $I_4 = I_2 = I_{out}$. Thus we can substitute the collector currents in eqn. (3.3) by I_{in1} , I_{in2} and I_{out} :

$$I_{in2} \cdot I_{in1} = I_{out} \cdot I_{out},$$

so $I_{out} = \sqrt{I_{in1} \cdot I_{in2}}$ (since I_{out} , being a collector current, must be positive). That means, the network “computes” the geometric mean of the two inputs.

Example 3.3. As an example for a network with several translinear loops, consider the network of Figure 3.4.

Transistors Q_1, Q_2, Q_6, Q_5 form a translinear loop. The according equation is

$$I_1 \cdot I_5 = I_2 \cdot I_6. \quad (3.4)$$

Another translinear loop consists of transistors Q_3, Q_4, Q_7, Q_6 . This gives

$$I_3 \cdot I_6 = I_4 \cdot I_7. \quad (3.5)$$

By Kirchhoff's Current Law and our neglect of base currents, we can rewrite eqn. (3.4) and eqn. (3.5) as

$$\begin{aligned} u_1 \cdot u_1 &= y^+ \cdot (y^+ + y^-), \\ y^- \cdot (y^+ + y^-) &= u_2 \cdot u_2. \end{aligned}$$

3. Translinear Network Theory

A little computation reveals that

$$y^+ - y^- = \frac{u_2^2 - u_1^2}{\sqrt{u_1^2 + u_2^2}}.$$

If we apply sinusoidal inputs with a 90° phase shift, like

$$\begin{aligned} u_1 &= |a \sin t|, \\ u_2 &= |a \cos t|, \end{aligned}$$

with a fixed $a \in \mathbb{R}$, then the differential output becomes

$$y^+ - y^- = a \cos 2t,$$

that is, the network shows a frequency doubling behaviour for these inputs.

So every translinear loop leads to an equation of the form of eqn. (3.2). Summarizing the translinear principle in words:

In a loop of base-emitter branches of transistors with the same thermal voltage and the same saturation current, with an equal number of forward and backward branches, the product of collector currents of the transistors whose base-emitter branches are forward in the loop is equal to the product of collector currents of the transistors whose base-emitter branches are backward in the loop.²

3.1.2. Motivation for a Catalog of Topologies

The following properties of static translinear (STL) networks can be observed from the examples of the preceding subsection:

- STL networks can be described in terms of currents by systems of polynomial equations.
- In such a system, no continuous parameters occur. This is due to the fact that U_T and I_S vanish from the equations as soon as the STL principle is applied.
- The topology of translinear networks satisfies strong constraints. One of these constraints is the condition that the number of forward and backward branches in a translinear loop be the same. Another constraint concerns the connection of collectors; it will be considered in Subsection 3.3.2.

²This is the author's version of many similar formulations of the translinear principle as found in the literature [Gil68, pp. 364–365], [See88, p. 9], [Gil96, p. 107], [Min97, p. 6].

3. Translinear Network Theory

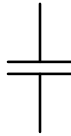
The second property means that the behavior of a STL network is already fixed by the network topology. In particular, there is only a finite number of different STL networks when the number of transistors is bounded!

Together with the third property, which says that the number is not only finite but also “not too large”, this observation has inspired the idea of a complete catalog of “small” STL networks. If along with each network appropriate equations are stored, such a catalog can serve as a design tool in an obvious way: When the designer is in search for a circuit with a given desired behavior, she or he can simply run through the catalog to find a network whose equations match that behavior.

Chapter 5 of this thesis is about the details of the usage of such a catalog.

3.1.3. The Dynamic Translinear Principle

For so-called dynamic translinear circuits, another circuit element comes into play: The capacitance. The symbol for a capacitance looks like this:



An ideal capacitance has the model equation

$$I_{\text{cap}} = C\dot{V}_{\text{cap}}, \quad (3.6)$$

where of course I_{cap} denotes the current through and V_{cap} the voltage across the element, and the dot is used to denote time derivative. The device parameter $C > 0$ is the **capacity**.

Consider a loop containing one capacitance and one or more base-emitter branches of transistors, as in Figure 3.5. This time, the branch orientations do not matter.

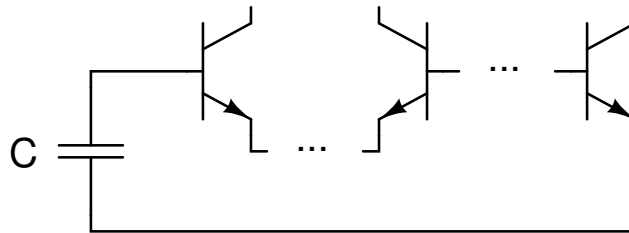


Figure 3.5.: A dynamic translinear loop.

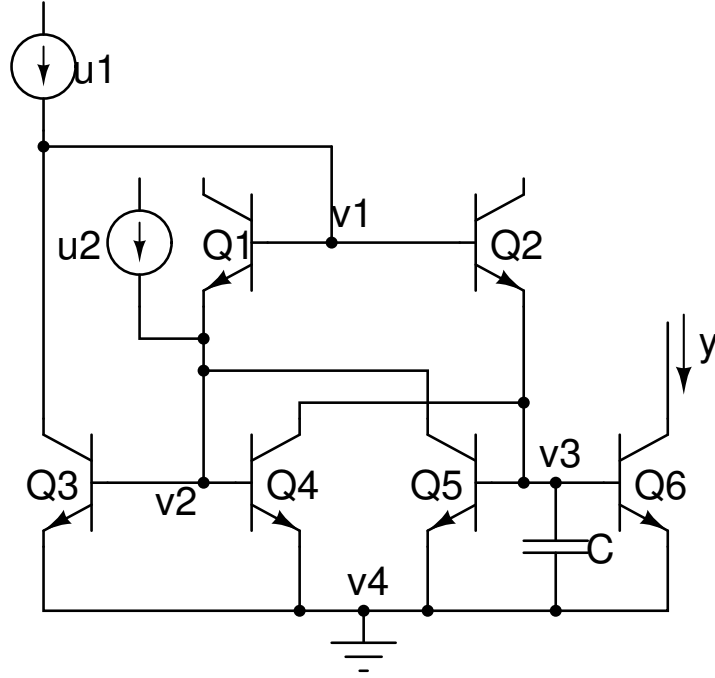


Figure 3.6.: A translinear integrating network. [See90]

KVL for such a **dynamic translinear** (DTL) loop is

$$V_{\text{cap}} = \sum_{j=1}^l \pm V_j, \quad (3.7)$$

where the signs depend on the branch orientations. From eqn. (3.1) we deduce that

$$\dot{V}_j = U_T \frac{\dot{I}_j}{I_j}$$

for $j = 1, \dots, l$, so eqn. (3.6) and the differentiation of eqn. (3.7) yield

$$\frac{1}{C} I_{\text{cap}} = U_T \sum_{j=1}^l \pm \frac{\dot{I}_j}{I_j}. \quad (3.8)$$

In short, the dynamic translinear principle says that for every DTL loop, eqn. (3.8) holds.

Example 3.4. Figure 3.6 shows a translinear integrating network. The (static) translinear loop equations for this network are

$$I_1 \cdot I_4 = I_2 \cdot I_5, \quad (3.9)$$

$$I_3 = I_4, \quad (3.10)$$

$$I_6 = I_5. \quad (3.11)$$

3. Translinear Network Theory

There is a dynamic translinear loop consisting of the capacitance and, say, Q_6 . (One could as well consider the loop consisting of the capacitance and Q_5 or the loop consisting of the capacitance, Q_2 , Q_1 and Q_4 .) The according equation is

$$\frac{1}{C}I_{\text{cap}} = U_T \frac{\dot{I}_6}{I_6}. \quad (3.12)$$

The node equations according to KCL are

$$I_3 = u_1 \quad \text{for } v_1, \quad (3.13)$$

$$I_5 = I_1 + u_2 \quad \text{for } v_2, \quad (3.14)$$

$$\text{and} \quad I_4 + I_{\text{cap}} = I_2 \quad \text{for } v_3. \quad (3.15)$$

I_1, \dots, I_5 and I_{cap} can be eliminated from eqns. (3.9) to (3.15), yielding

$$(u_2 - I_6) \cdot u_1 = (u_1 + CU_T \frac{\dot{I}_6}{I_6}) \cdot I_6,$$

which can be simplified to

$$u_2 \cdot u_1 = CU_T \dot{I}_6.$$

If we assume the input u_2 to be a constant scaling factor, we see that the input u_1 is proportional to the time derivative of the output $y = I_6$. So indeed, the network effectively performs integration.

Although dynamic translinear networks have important applications, this thesis is mainly about STL networks, and capacitances won't occur very often. In particular, the implementation of a topological catalog as a synthesis tool, which has been produced in the framework of this thesis, is restricted to STL networks.

3.2. Translinear Decomposition of Polynomials

We use the term “translinear decomposition” to denominate the process (or the result) of finding a way of writing a polynomial $f \in \mathbb{Q}[x_1, \dots, x_n]$ as an algebraic expression which can be interpreted as one or more translinear loop equations. We think of the variables x_1, \dots, x_n as the inputs and outputs of a network to be designed, and of f as an implicit description of the network's desired behavior. Translinear decomposition is an important step of the design trajectory for translinear networks as described by Mulder et. al. [MSvdWvR99].

If one is only interested in networks with only one translinear loop, translinear decomposition amounts to finding a way of writing f in the form

$$f = L_1 \cdot \dots \cdot L_r - M_1 \cdot \dots \cdot M_r,$$

3. Translinear Network Theory

where L_i and M_i are linear combinations of x_1, \dots, x_n . An algorithm for translinear decomposition in the 1-loop case is included in the work of Mulder et. al. [MSvdWvR99, pp. 91–107]. The author has developed an alternative algorithm [Ils02] and has compared both algorithms using implementations in MATHEMATICA.

In the general case of several translinear loops, translinear decomposition is much more complicated: Given f , we have to look for “translinear polynomials”

$$\begin{aligned} f_1 &= L_{11} \cdot \dots \cdot L_{1r_1} - M_{11} \cdot \dots \cdot M_{1r_1}, \\ &\vdots \\ f_s &= L_{s1} \cdot \dots \cdot L_{sr_s} - M_{s1} \cdot \dots \cdot M_{sr_s} \end{aligned}$$

(corresponding to s translinear loops) in the enlarged polynomial ring

$$\mathbb{Q}[x_1, \dots, x_n, x_{n+1}, \dots, x_{n+s-1}]$$

such that $f_1 = \dots = f_s = 0$ implies $f = 0$ for any given set of real values for x_1, \dots, x_{n+s-1} . Here the terms L_{ij} and M_{ij} denote linear combinations of the variables x_1, \dots, x_{n+s-1} .

The condition “ $f_1 = \dots = f_s = 0$ implies $f = 0$ ” can be ensured algebraically by choosing f_1, \dots, f_s in such a way that f lies inside the ideal $\langle f_1, \dots, f_s \rangle$, that is, in such a way that there exist $h_1, \dots, h_s \in \mathbb{Q}[x_1, \dots, x_{n+s-1}]$ with

$$f = h_1 f_1 + \dots + h_s f_s. \quad (3.16)$$

Note that since f contains only x_1, \dots, x_n , the remaining variables $x_{n+1}, \dots, x_{n+s-1}$ must cancel on the right hand side of eqn. (3.16).

No good algorithms are known for translinear decomposition in the general case (also called *parametric decomposition*), and it doesn’t seem probable that a satisfactory algorithmic solution can be found, since the problem is in some sense “the wrong way around” compared to classical problems of computer algebra. Also, there are so many degrees of freedom that one can expect a very large number of solutions among which it would be complicated to recognize “good” ones.

Another problem that arises when employing the “translinear decomposition” design trajectory is that it is not clear, once a suitable decomposition is found, how to make the collector currents in fact equal the linear combinations L_{jk} and M_{jk} .

Still, it seems worthwhile to research about the algebraic problem of translinear decomposition. However, due to the problems mentioned, the design approach of this thesis avoids translinear decomposition.

3.3. The Topology of Translinear Networks

In this section, we give a precise formulation of what a “translinear network” is in terms of graph theory. It will be the basic mathematical model of a translinear circuit’s topology and consists essentially of a strict formulation of the constraints which the topology of translinear networks has to obey. Although these constraints have all been known before (their identification is due to E. Seevinck [See88]), their translation into strict mathematics is new.

The precise mathematical formulation is necessary for the specification of the combinatorial task of compiling complete lists of topologies, considered in Chapter 4.

It has been developed in collaboration with E. J. Roebbers and has been published earlier [Ils04, IR04]. The successful application (see Chapter 6) of the techniques of this thesis prove that the topological model fits very well with the industrial needs.

3.3.1. Translinear Digraphs

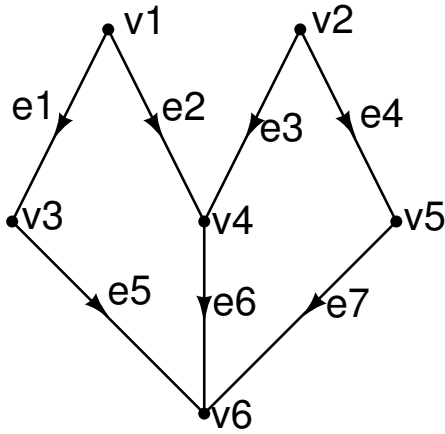
Translinear digraphs are the mathematical objects that are used to represent the core structure of a translinear network, the structure consisting of the translinear loops.

Definition 3.5. A **translinear digraph** is a digraph G satisfying the following properties:

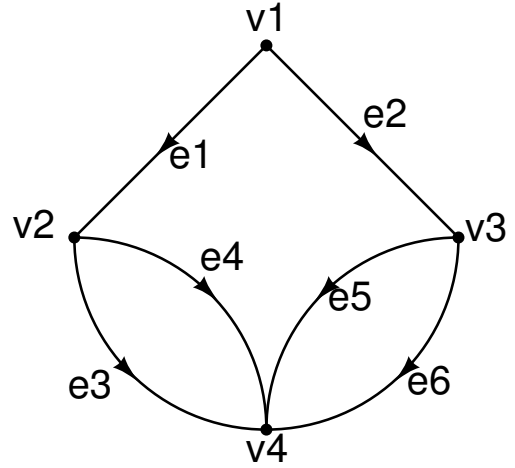
1. Every loop of G has as many forward branches as backward branches.
2. G is biconnected.

One should think of a translinear digraph G as the digraph formed by the base-emitter branches of a translinear network, that is, the branches of G are in 1-to-1-correspondence with the transistors of the network, and for each branch e , $\text{tail}(e)$ corresponds to the base node and $\text{head}(e)$ to the emitter node of the respective transistor.

Figure 3.7 shows the translinear digraphs of the example networks from Section 3.1.



(a) The translinear digraph of the frequency doubling network (see Figure 3.4).



(b) The translinear digraph of the integrating network (Figure 3.6).

Figure 3.7.: Two translinear digraphs. The nodes carry the same names as in the corresponding networks; branch e_j in the digraphs corresponds to transistor Q_j in Figure 3.4 or Figure 3.6, respectively.

Condition 1 in Definition 3.5 obviously reflects the main requirement on translinear loops as presented in Section 3.1. The reason for including Condition 2 into the definition is that the loops of different biconnected components of the base-emitter digraph are decoupled, so we can consider the corresponding sub-networks separately.

The concept of a translinear digraph as the core of a translinear network was introduced by E. Seevinck [See88], although he concentrated on undirected graphs.³ (Seevinck's definition differs from the one given here in some more respects.)

It turns out that condition 1 of Definition 3.5 has a nice reformulation, as expressed by the following theorem:

Theorem 3.6. *Let G be a digraph. The following two statements are equivalent:*

1. *Every loop of G has as many forward arcs as backward arcs.*
2. *There exists a map $r : V(G) \rightarrow \mathbb{Z}$ such that*

$$\forall e \in E(G) : r(\text{tail}(e)) = r(\text{head}(e)) + 1.$$

³This is the reason that the term "translinear graph" is found very often in the literature, whereas "translinear digraphs" have not found much attention hitherto.

3. Translinear Network Theory

Proof. First assume there exists a map r as in statement 2. Consider any loop $W = (v_0, e_1, v_1, \dots, e_l, v_l)$, $v_l = v_0$ of G . By assumption,

$$r(v_j) = r(v_{j-1}) - \mu(W, e_j)$$

for all $j = 1, \dots, l$. Inductively it follows that

$$r(v_l) = r(v_0) - \sum_{j=1}^l \mu(W, e_j)$$

and, because $v_l = v_0$,

$$\sum_{j=1}^l \mu(W, e_j) = 0.$$

Since $\mu(W, e_j) = \pm 1$ for all $j = 1, \dots, l$, the latter is a sum of as many 1's as -1's, which just means that W contains as many forward arcs as backward arcs.

For the other direction of the proof, assume that every loop contains as many forward arcs as backward arcs. In terms of incidence indices, this means

$$\sum_{e \in E(W)} \mu(W, e) = 0 \tag{3.17}$$

for every loop W . It follows that eqn. (3.17) also holds if W is any cycle.

Construct a map r with the stated property in the following way: For each connected component C of G , pick an arbitrary vertex $v_C \in V(C)$. Then, for each vertex $v \in V(C)$, fix a walk P from v_C to v and define

$$r(v) := - \sum_{e \in E(P)} \mu(P, e).$$

It is now necessary to show that this definition is independent of the chosen walk P . So, let P' be another walk from v_C to v . Then $P \star \overleftarrow{P'}$ is a cycle and

$$\begin{aligned} 0 &= \sum_{e \in E(P \star \overleftarrow{P'})} \mu(P \star \overleftarrow{P'}, e) \\ &= \sum_{e \in E(P)} \mu(P, e) - \sum_{e \in E(P')} \mu(P', e), \end{aligned}$$

in particular

$$- \sum_{e \in E(P)} \mu(P, e) = - \sum_{e \in E(P')} \mu(P', e),$$

which shows that $r(v)$ is indeed well-defined.

Applying this construction to every connected component C of G , the map r indeed gets the desired property: Let $e_0 \in E(C)$ be any branch of some component C . If W

3. Translinear Network Theory

is a walk from v_C to $\text{tail}(e_0)$, $W' := W \star e_0$ is a walk from v_C to $\text{head}(e_0)$, and by the definition of r ,

$$\begin{aligned}
 r(\text{head}(e_0)) &= - \sum_{e \in E(W')} \mu(W', e) \\
 &= - \left(\sum_{e \in E(W)} \mu(W', e) \right) - \mu(W', e_0) \\
 &= - \left(\sum_{e \in E(W)} \mu(W, e) \right) - 1 \\
 &= r(\text{tail}(e_0)) - 1.
 \end{aligned}$$

□

It is clear that if a map r as in the second statement of Theorem 3.6 exists, so does a map r_0 that fulfills the same condition as well as the additional property

$$\min_{v \in V(G)} r_0(v) = 0. \tag{3.18}$$

(Simply define $r_0(v) := r(v) - \min_{v' \in V(G)} r(v')$.) The nodes can then be partitioned into “levels” or “layers”, such that a branch always points from one layer to the next lower layer:

$$V = V_0 \dot{\cup} V_1 \dot{\cup} \dots \dot{\cup} V_R,$$

where $V_j = \{v \in V \mid r_0(v) = j\}$ and $R := \max_{v \in V} r_0(v)$. This is illustrated in Figure 3.8.

Example 3.7. In Figure 3.7(a), $R = 2$, $V_0 = \{v_6\}$, $V_1 = \{v_3, v_4, v_5\}$ and $V_2 = \{v_1, v_2\}$.

In Figure 3.7(b), $R = 2$, $V_0 = \{v_4\}$, $V_1 = \{v_2, v_3\}$ and $V_2 = \{v_1\}$.

Definition 3.8. We call a digraph fulfilling one and thus both conditions of Theorem 3.6 a **layered digraph**.

We see that a translinear digraph is nothing but a biconnected layered digraph.

For any connected layered digraph, the additional property of eqn. (3.18) makes r_0 unique.

Definition 3.9. Let G be a connected layered digraph. For a node $v \in V(G)$, we call the integer $r_0(v)$ the **rank** of v , and also denote it by $\text{rank}(v)$.

In other words, the rank of a node is the index of the layer it belongs to.

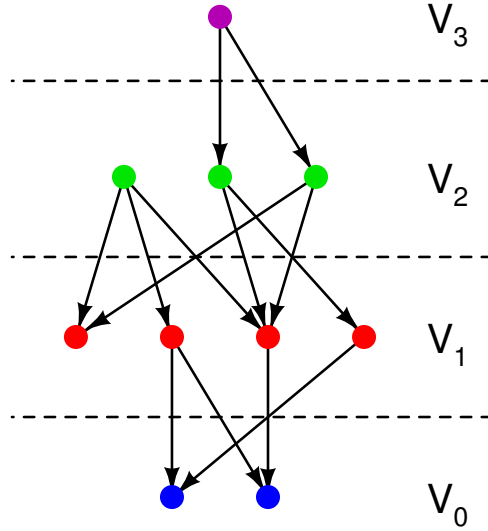


Figure 3.8.: layers of a translinear digraph

What is crucial about the layers of the translinear digraph of a network is that the rank of a node corresponds nicely to a certain range of the electric potential the node will have in an actual circuit. This is because the voltage drop along a base-emitter branch is always much larger than the swing of potential at one particular node.

Since the rank is a node invariant, it also helps a lot in the classification of translinear digraphs, as will become apparent in Chapter 4.

3.3.2. Connection of Collectors

The previous subsection dealt with the base-emitter connectivity of a translinear network, which can be encoded in a translinear digraph. The main piece of information that is furthermore needed to describe a complete network is where to connect the collectors to.

Let e be a branch of a translinear digraph G . The collector of the transistor corresponding to e can only be connected to a node v of G if $\text{rank}(v) > \text{rank}(\text{head}(e))$. The reason is that $\text{head}(e)$ is the emitter node, and the collector needs a higher potential than the emitter. E. Seevinck considered this condition in the construction of his “T-matrices” [See88].

We denote the collector node of a transistor corresponding to a branch e of the translinear digraph by $C(e)$.

Example 3.10. For the integrating network (Figure 3.6), we have $C(e_3) = v_1$, $C(e_4) = v_3$ and $C(e_5) = v_2$ (using branch names as in Figure 3.7(b)).

3. Translinear Network Theory

But a collector does not necessarily need to be connected to a node of the translinear digraph. It can also serve as a (current-mode) output of the network, or it can be connected directly to a voltage supply. We will express this by $C(e) = v_{\text{ext}}$, imagining v_{ext} as an additional node outside of G .

Example 3.11. In Figure 3.6/Figure 3.7(b), $C(e_1) = C(e_2) = C(e_6) = v_{\text{ext}}$.

In summary:

A translinear network topology is specified by

- a translinear digraph G
- and a map $C : E(G) \rightarrow V(G) \cup \{v_{\text{ext}}\}$, where C has the property that for each $e \in E(G)$, either $C(e) = v_{\text{ext}}$ or

$$\text{rank}(C(e)) > \text{rank}(\text{head}(e)). \quad (3.19)$$

We identify

a branch e	with	a transistor,
tail(e)	with	the transistor's base node,
head(e)	with	the transistor's emitter node,
and $C(e)$	with	the transistor's collector node.

We call the map C the **collector assignment** of the network. (The notion of a collector assignment will be refined in Section 3.4.)

Example 3.12. We can describe the frequency doubling network of Figure 3.4 by the pair (G, C) , where G is the translinear digraph depicted in Figure 3.7(a) and C is the collector assignment

$$\begin{aligned} C(e_1) &= v_1, \\ C(e_2) &= v_{\text{ext}}, \\ C(e_3) &= v_{\text{ext}}, \\ C(e_4) &= v_2, \\ C(e_5) &= v_3, \\ C(e_6) &= v_4, \\ C(e_7) &= v_5. \end{aligned}$$

Remark 3.13. Note that in the latter example, $C(e_j) = \text{tail}(e_j)$ for $j = 1, 4, 5, 6$, meaning that the collector is connected to the same node as the base of the respective transistor. In such cases of diode-like transistor usage, the condition expressed by eqn. (3.19) is “automatically” satisfied, since by the definition of the rank of a node,

$$\text{rank}(\text{tail}(e)) = \text{rank}(\text{head}(e)) + 1.$$

The information given by a translinear digraph G and a collector assignment C is already a fairly complete description of a translinear network. In particular, a netlist for simulation or symbolic analysis of the network can be set up, if just some necessary “interface” information is added, for instance the connection of independent current sources which represent inputs of the network. These interfacing issues will be considered in Section 3.4.

3.3.3. Insertion of Capacitances

The considerations of the preceding subsection are valid for STL networks as well as for DTL networks. But while the topology of a STL network is sufficiently modeled by a translinear digraph and a collector assignment, we must make one addition for DTL networks: How are capacitances allowed to be connected in the network?

The answer is quite simple: A capacitance can be inserted between any pair of nodes of the translinear digraph. In Figure 3.6, for example, the capacitance branch is between node v_3 and the ground node v_4 , which are both nodes of the underlying translinear digraph (see Figure 3.7(b)).

Thus, a DTL network topology is given by a triple (G, C, E_{cap}) of a translinear digraph G , a collector assignment C on G , and a subset E_{cap} of the set of 2-element subsets of G . (See also [IR04].) E_{cap} then consists of those pairs of nodes where a capacitance is connected inbetween.

3.4. The Interface of a Translinear Network

This section addresses the question of what should be considered as the “interface” of a translinear network, i. e. of how inputs are applied to and outputs are supplied by the network.

Since the introduction of the translinear principle by Barrie Gilbert [Gil75] it is clear that inputs and outputs of a translinear network are in current mode.

3.4.1. Outputs

All collectors which are designated “external” by the collector assignment C (i. e., the collectors of those transistors for which $C(e) = v_{\text{ext}}$) can be used as outputs of the network. But not only the current of a single collector, also sums or differences of them can be considered as outputs. To be general, we consider a single symbolic output y of a network which is a sum of positive and negative collector currents:

$$y = \sum_{C(e)=v_{\text{ext}}} \sigma(e)x_e, \quad (3.20)$$

3. Translinear Network Theory

where x_e denotes the collector current of the transistor corresponding to branch e and $\sigma(e) \in \{-1, 0, 1\}$.

Example 3.14. For the frequency doubling network (Figure 3.4), $\sigma(e_2) = 1$ and $\sigma(e_3) = -1$. For the integrating network (Figure 3.6), $\sigma(e_1) = \sigma(e_2) = 0$ and $\sigma(e_6) = 1$.

To avoid the usage of σ , we will henceforth work with the following refined concept of a collector assignment: From now on, a collector assignment will be a map

$$C : E(G) \rightarrow V(G) \dot{\cup} \{v_{\text{out}+}, v_{\text{out}-}, v_{\text{void}}\}$$

such that for every $e \in E(G)$:

$$C(e) \in \{v_{\text{out}+}, v_{\text{out}-}, v_{\text{void}}\} \text{ or } \text{rank}(C(e)) > \text{rank}(\text{head}(e)).$$

Thus, the symbolic output of a network will be (compare to eqn. (3.20)):

$$y = \sum_{C(e)=v_{\text{out}+}} x_e - \sum_{C(e)=v_{\text{out}-}} x_e. \quad (3.21)$$

3.4.2. Inputs and Ground Node Selection

In principle, an independent input current i_v can be applied to any node v of the translinear digraph G . The only restriction is that the sum of all input and output currents (the latter are now all collector currents with $C(e) \in \{v_{\text{out}+}, v_{\text{out}-}, v_{\text{void}}\}$) must always be zero, which cannot be satisfied if independent currents are prescribed for all nodes of G . (In this case, the modelling assumptions about the transistors are not valid anymore.) Therefore, one has to select one particular node v_0 of G that serves as a “valve” whose “dependent input” current results from values of the “true” inputs and of the outputs. (One should regard this dependence in terms of KCL for the ground node.)

For convenience, we will always choose this “valve” node as the “reference” or “ground” node of the circuit, thus there will be no distinction between “valve node” and “ground node”, and we simply assume that we have an independent current source connected to each node of G except one, which we call the ground node and denote by v_0 . We call all other nodes “input nodes”.

In the examples we have seen so far, most of the nodes of the translinear digraph have no current source connected to them. This amounts to an independent input which happens to be a constant zero and is not to be confused with the role of the ground node!

Example 3.15. For the frequency doubling network (Figure 3.4): $v_0 = v_6$; $i_{v_1} = u_1$, $i_{v_2} = u_2$, $i_{v_3} = i_{v_4} = i_{v_5} = 0$.

Example 3.16. For the integrating network (Figure 3.6): $v_0 = v_4$; $i_{v_1} = u_1$, $i_{v_2} = u_2$, $i_{v_3} = 0$.

A triple (G, C, v_0) of a translinear digraph G , a collector assignment C (in the refined sense) and a ground node v_0 is a network description which is complete in the sense that the network equations can entirely be set up. The resulting system of polynomial equations will be studied in Section 5.1. That system contains all node equations but the one of v_0 . For this reason, the current into the collector of the transistor corresponding to a branch e with $C(e) = v_0$ does not occur in the network equations. The same is true if $C(e) = v_{\text{void}}$, while in all other cases, the collector current belonging to e does affect the system: either in the node equation of $C(e) \in V(G) \setminus \{v_0\}$ or, if $C(e) \in \{v_{\text{out}+}, v_{\text{out}-}\}$, in the output equation.

Hence, for the system of network equations, it does not matter whether $C(e) = v_{\text{void}}$ or $C(e) = v_0$ for a branch e . To avoid redundant entries in our catalog, we do not allow $C(e) = v_0$ for any e . The simplest possibility to guarantee $C(e) \neq v_0$ for all e is to choose $v_0 \in V(G) \setminus \text{img}(C)$.

We arrive at the following precise formal definition of a translinear network:

Definition 3.17. A **(static) formal translinear network** is a triple $N = (G, C, v_0)$ of a translinear digraph G , a map $C : E(G) \rightarrow V(G) \dot{\cup} \{v_{\text{out}+}, v_{\text{out}-}, v_{\text{void}}\}$ and a node $v_0 \in V(G) \setminus \text{img}(C)$, such that for every $e \in E(G)$:

$$C(e) \in \{v_{\text{out}+}, v_{\text{out}-}, v_{\text{void}}\} \quad \text{or} \quad \text{rank}(C(e)) > \text{rank}(\text{head}(e)).$$

We call C the **collector assignment of N** and v_0 the **ground node of N** . The **number of transistors of N** is the number of branches of G .

3.5. Topologies for 4-terminal MOS transistors

Next to bipolar transistors, subthreshold MOS transistors can be employed as “translinear device”.

In the preceding sections, we frequently used a terminology that is adopted from the bipolar case. Speaking of MOS transistors, we should replace “base” by “gate”, “emitter” by “source” and “collector” by “drain”. But in addition to these three terminals, several authors have begun to use the “back gate” or “bulk” terminal of MOS transistors as an independent fourth terminal, instead of short-circuiting it “by law” with the source terminal [Ser05, MvdWSvR95, AB96, SGLBA99].

The topological concepts developed in the preceding sections are based on 3-terminal transistor devices. This section gives some ideas to adapt the concepts for 4-terminal MOS devices. However, a coherent description as for the 3-terminal case is not achieved.

Our proposals are based on the “general translinear principle for subthreshold MOS transistors” by Serrano-Gotarredona, Linares-Barranco and Andreou [SGLBA99]. We restrict to subthreshold MOS transistors employed in forward region.⁴ For examples, we refer to the literature mentioned above.

⁴We did the same for bipolar transistors by using eqn. (3.1) as model equation.

3. Translinear Network Theory

MOS translinear loops consist either of gate-source branches or of bulk-source branches. Analogously to the translinear digraph, which is the “base-emitter” digraph of bipolar translinear networks, we can consider a gate-source digraph G_{gate} and a bulk-source digraph G_{bulk} of a MOS translinear network. G_{gate} and G_{bulk} share a common node set

$$V := V(G_{\text{gate}}) = V(G_{\text{bulk}}),$$

and furthermore, there is bijection

$$h : E(G_{\text{gate}}) \rightarrow E(G_{\text{bulk}})$$

identifying branches representing the same transistor, such that

$$\text{tail}(e) = \text{tail}(h(e))$$

for each $e \in E(G_{\text{gate}})$. (Since both tails are to be identified with the source of the transistor.)

According to [SGLBA99], it is not necessary to impose any loop condition similar to the one for the bipolar case (“as many forward as backward branches”) to G_{gate} or G_{bulk} . The biconnectedness condition applies to the “superposition” digraph

$$(V, E(G_{\text{gate}}) \dot{\cup} E(G_{\text{bulk}}), \iota).$$

The examples in [SGLBA99] show that neither G_{gate} nor G_{bulk} can be assumed to be biconnected by itself, in fact, they cannot even be assumed to be connected.

For connecting the drain terminals, one should require that they are provided with a higher potential than the corresponding source terminal. However, lacking layers of G_{gate} or G_{bulk} , we cannot use a convenient formal condition, like eqn. (3.19) for the bipolar case, to express this requirement. In the example circuits known to the author, all drain terminals are either used as outputs or are short-circuited with the gate terminal of the respective transistor.

4. Combinatorial Generation of Translinear Networks

The preceding chapter provides a way to regard translinear networks as formal combinatorial objects. This chapter describes techniques to generate complete lists of these formal objects, in order to compile catalogs of translinear networks which are exhaustive in the sense that they contain all possible topologies.

Synthetic lists of translinear graphs and translinear digraphs have been produced before [See88, Wie93]. However, the idea to list complete network descriptions is new.

The first section of this chapter introduces a common method in combinatorics to list graphs or other combinatorial objects. Since the core structures of translinear networks are translinear digraphs, Section 4.3 describes how to specialise the method for this particular class of digraphs, prepared by Section 4.2 on the generation of general digraphs. Sections 4.4 and 4.5 then deal with the exhaustive generation of collector assignments and complete formal networks for a given translinear digraph.

4.1. Orderly Generation

This section describes orderly generation, a method for exhaustive generation of combinatorial objects. It was developed mainly by R. Read [Rea78]. It will be applied to the generation of formal translinear networks in the following sections.

Subsection 4.1.1 introduces the general class of problems which orderly generation applies to. Subsection 4.1.2 then covers the basic ideas of orderly generation, and in Subsection 4.1.3 the main type of applications of orderly generation is presented.

4.1.1. Cataloging Problems

The term “catalog” is used here to refer to a complete but redundance-free list of combinatorial structures such as elementary ones like sequences, permutations or partitions, but also more complex ones like graphs in several variants (simple graphs, multigraphs, digraphs, trees, colored graphs, etc.), designs [Dem97, And90, AK92, Col96, BJJ99] or linear codes [BFK⁺98, in particular chapter 3].¹

¹Of course, the literature provides plenty of more examples of combinatorial structures which a catalog makes sense of. In fact, structure is brought into the vast range of examples by a well-developed the-

4. Combinatorial Generation of Translinear Networks

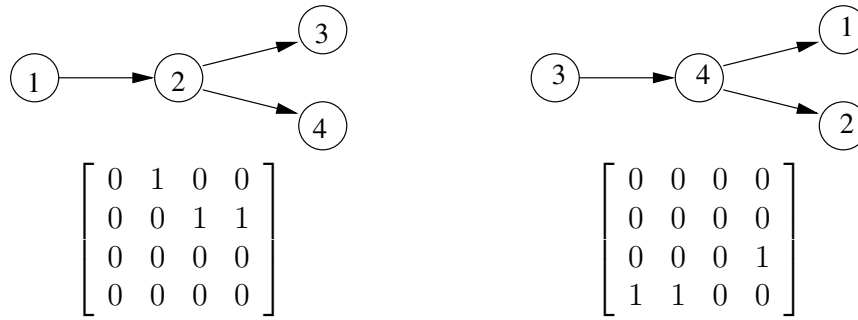


Figure 4.1.: Two labeled digraphs and their adjacency matrices.

Catalogs of these structures are useful in many respects: In pure mathematics, they can serve as a source of examples, especially for testing new conjectures. Also, classification problems in many mathematical disciplines often boil down to discrete cataloging problems. In practical applications, catalogs allow to get hands on error-correcting linear codes, designs of agricultural experiments, isomers of a molecule, or, being the motivation of the present work, possible topologies of electrical networks.

Main references for cataloging techniques, which can also be paraphrased as *generation of structures*, are [Lau93, Lau99, GLM97, Rea78]. Related but not to be confused with generation is *enumeration* (counting) of structures, treated by [Red27, PR87, KT83, GJ83, dB64]. Both generation and enumeration are dealt with in [Ker99] or [KS99].

Except for the most simple structures, it is a nontrivial task to produce a catalog. The difficulties will become apparent after the following general formulation of a cataloging problem.

Assume that for each $b \in \mathbb{N}$, we have a finite set L_b whose elements are easily digitally represented and also easily listed in the sense that we can produce a list of all elements of L_b in a time that is proportional to their number $|L_b|$. Furthermore, assume $L_b \cap L_{b'} = \emptyset$ for $b \neq b' \in \mathbb{N}$ and denote $L := \bigcup_{b=0}^{\infty} L_b$. We call the elements of L_b the **labeled structures of size b** .

In the case of digraphs, for example, we take a fixed set of nodes V and let L_b be the set of digraphs with node set V and exactly b branches. Adjacency matrices are a convenient digital representation of digraphs, and we can think of L_b as those matrices whose total sum of entries is b .

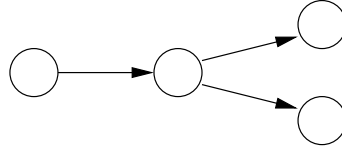
Figure 4.1 shows two digraphs and their adjacency matrices. Since the matrices differ, we have two different elements of L_b .

Unfortunately, the labeled structures are not yet what we are really interested in. Rather, we are interested in **unlabeled structures** which are isomorphism classes of the labeled structures. Since the two digraphs in Figure 4.1 are clearly isomorphic, we do not want

ory of so-called **species** representing the types of combinatorial structures. [Ehr65, Joy81, BLL98], see also [Ker99, pp. 1–20].

4. Combinatorial Generation of Translinear Networks

to have both of them in a list of “all digraphs with 4 nodes and 3 branches”, rather we want to have only one representative looking like this:



The example makes quite clear where the terminology “labeled” and “unlabeled” comes from.

In general, we have an equivalence relation \sim on L_b , where $A \sim B$ is to be interpreted as “ A and B differ only by their labelings” or “ A and B are isomorphic”.² Thus, an **unlabeled structure of size b** is an element of L_b / \sim .

In the case of digraphs with n nodes and b branches, the equivalence relation in terms of adjacency matrices is the one induced by the action of S_n consisting of simultaneous row and column permutations: Two $n \times n$ matrices $A, B \in L_b$ are “isomorphic” if and only if there is a permutation σ which, when applied simultaneously to rows and columns, transforms A into B :

$$A \sim B \iff \exists \sigma \in S_n : \text{For all } i, j = 1, \dots, n : B_{ij} = A_{\sigma(i)\sigma(j)}.$$

(In Figure 4.1 above, $\sigma = (1\ 3)(2\ 4)$ transforms the two matrices into each other. σ also permutes the node labels of the two digraphs in the appropriate way.)

The example of digraphs is presented here because digraphs are of course the structures we are specifically interested in for our purposes; further examples of labeled and unlabeled structures are nicely presented in the first chapters of [Ker99]. Quite an interesting example is the one of linear codes, where equivalence is isometry of codes, i.e. equivalent codes are guaranteed to show the same error-correcting behavior.

In general, the best way to represent an unlabeled structure digitally is to represent it by one of the labeled structures it is made up of. That means that we can define a catalog as a subset $U \subset L$ which has the property that for every $A \in L$ there is a unique $B \in U$ such that $A \sim B$. In other words, U should contain exactly one element from each isomorphism class. Such a set U is called a **complete system of representatives** or a **transversal** of L / \sim .

It is worth to point out that in principle, there is of course no problem to list unlabeled structures *completely*: One can just list the labeled structures. Every unlabeled structure will be represented in the list at least once. The problem is that because in most cases it will be represented by quite a lot of labeled structures, the list gets very redundant

²In more precise notation, one would use \sim_b instead of \sim . Omitting the reference to b does not, however, cause any ambiguity. In fact, we can define \sim globally on $L = \bigcup_{b=1}^{\infty} L_b$ by specifying that $A \sim B$ should imply that A and B are of the same size.

and much too long, so that even when one just wants a complete list and redundancy doesn't disturb by itself, the reduction of complexity obtained by eliminating isomorphs can be a great improvement of efficiency.

The naive approach for generating a transversal of L_b / \sim is expressed by the following pseudocode:

Algorithm 1 generateTransversal

```

1:  $U_b := \emptyset$ 
2: for all  $x \in L_b$  do
3:   if there is no  $u \in U_b$  with  $x \sim u$  then
4:      $U_b := U_b \cup \{x\}$ 
5:   end if
6: end for
7: return  $U_b$ 

```

This algorithm is very slow because the isomorphism test $u \sim x$, which is in most cases very expensive, is performed very often. In the following subsection it will be shown how this can be avoided.

4.1.2. Canonicity and Augmentation

The basic idea to avoid the isomorphism test of Algorithm 1 is very intuitive:

If we find some definition of “canonicity”, a property that exactly one element of each \sim -class possesses, we can simply test all labeled structures for this property and thus generate the set of “canonical” elements of L_b , which is then our transversal of L_b / \sim :

Algorithm 2 generateCanonicalTransversal

```

1:  $U_b := \emptyset$ 
2: for all  $x \in L_b$  do
3:   if  $x$  is canonical then
4:      $U_b := U_b \cup \{x\}$ 
5:   end if
6: end for
7: return  $U_b$ 

```

This is a considerable improvement compared to Algorithm 1, because the so-far built U_b doesn't need to be searched through in each cycle.

Note that it does not matter for Algorithm 2 if we replace U_b by the complete U , this amounts to the same as successive generation of each U_b . In contrast to this, the splitting of U into the U_b 's is substantial for Algorithm 1, since in line 3 we would have to check all elements $u \in U$, instead of only $u \in U_b$, for being an isomorph of x .

4. Combinatorial Generation of Translinear Networks

An appropriate definition of canonicity is easily found: Usually, the set L of labeled structures is endowed with some natural total order, which comes from a numerical or lexicographical interpretation of the “labelings” and their digital representations. For example, one can imagine a 0-1-matrix (representing a simple labeled graph or digraph), read row by row, as a binary encoded integer. Thus, graphs are ordered by the integer ordering.

But once we have fixed a total order $<$ on L , we simply define the canonical representative of a \sim -class to be the maximal one with respect to $<$,

$$x \in L \text{ canonical} \quad :\iff \quad y < x \text{ for all } y \in L \text{ with } y \sim x \text{ and } y \neq x.$$

The details of the canonicity check in algorithm 2 of course depend on the definition of a canonical structure, but it can be expected that this test is as expensive as one isomorphism test. For instance, to test two digraphs with n nodes for isomorphism, in general one has to try the $n!$ bijections between their node sets. To test the adjacency matrix A of one n -node digraph for canonicity, one has to compare A with the $n!$ row- and column- permuted matrices $P_\sigma^{-1}AP_\sigma$, $\sigma \in S_n$, to find out whether A is maximal. (Here, P_σ denotes the permutation matrix of σ , defined by

$$(P_\sigma)_{ij} = \begin{cases} 1 & \text{if } \sigma(i) = j \\ 0 & \text{otherwise} \end{cases}$$

for $i, j = 1, \dots, n$.)

Thus it is further desirable to reduce the number of canonicity checks. This is possible if a suitable *augmentation* operation is at hand which produces structures of size $b + 1$ from a structure of size b . In most applications, plenty of different augmentations can be thought of. To make an augmentation really useful, it should satisfy the following requirement.

Definition 4.1. Let $U \subseteq L$ be a transversal of L/\sim . A **U -complete augmentation** is a map

$$\text{aug} : L \rightarrow \mathcal{P}(L)$$

such that for every $b \in \mathbb{N}$, $\text{aug}(x) \subseteq L_{b+1}$ for $x \in L_b$ and

$$\bigcup_{x \in U_b} \text{aug}(x) \supseteq U_{b+1},$$

where $U_b := U \cap L_b$.

An example of a quite versatile U -complete augmentation is given in the next subsection.

If aug is a U -complete augmentation, the following algorithm obviously computes the transversal U_{b+1} from U_b :

Algorithm 3 generateByAugmentation

```

1:  $U_{b+1} := \emptyset$ 
2: for all  $u \in U_b$  do
3:   for all  $x \in \text{aug}(u)$  do
4:     if  $x$  is canonical then
5:        $U_{b+1} := U_{b+1} \cup \{x\}$ 
6:     end if
7:   end for
8: end for
9: return  $U_{b+1}$ 

```

Algorithm 3 can be successively applied from $b = 0$ on (assuming $U_0 = L_0$ or something likewise trivial) to a desired size.

The fundamental difference compared to algorithms 1 and 2 is that in order to take advantage of the augmentation, we now really depend on the decomposition $U = \bigcup U_b$. The gain is that it is not necessary to run through all labeled structures of size b to generate the transversal U_b .

Remark 4.2. If the augmentation has the property that every structure of size $b + 1$ has a *unique* predecessor of size b , that is, if $\text{aug}(u) \cap \text{aug}(u') = \emptyset$ for $u \neq u'$, then we know for sure that in line 3 of Algorithm 3, x is not yet included in U_{b+1} . This makes the insertion of x into U_{b+1} easier to implement and faster. We will make use of this advantage throughout our particular applications.

A similar efficiency of insertion can be achieved if we choose the canonicity to be maximality with respect to some total order $<$, as proposed above, and furthermore the elements of $\text{aug}(x)$ are always given as a $<$ -ordered list: We can then store U_{b+1} as a list that we always keep $<$ -ordered as well, so that in line 3 of Algorithm 3, we only need to compare x with the last element of U_{i+1} to find out whether x is to be appended or not. This variant is the original orderly generation by Read [Rea78] and the best justification of its name.

4.1.3. Generation of Canonical Words

This subsection deals with the generation of canonical words, which is both a good example to illustrate the technique of orderly generation and also the basic tool which will be used in the following sections to produce catalogs of digraphs and networks.

Let M be a finite or countable set and let l be a positive integer. The labeled objects are **the words of length l over M** , that is,

$$L = M^l = \{(x_1, \dots, x_l) \mid x_j \in M\}.$$

4. Combinatorial Generation of Translinear Networks

Now let $\mathcal{G} \leq S_l$ be a permutation group and take \sim to be the equivalence relation induced by \mathcal{G} on M^l : For $x, y \in M^l$,

$$x \sim y \quad :\iff \quad \exists \sigma \in \mathcal{G} : y = x^\sigma,$$

where $x^\sigma := (x_{\sigma(1)}, \dots, x_{\sigma(l)})$ if $x = (x_1, \dots, x_l) \in M^l$.

Our goal is to apply orderly generation to produce a transversal of $M^l / \sim = M^l / \mathcal{G}$.

From now on, we assume that $M = \mathbb{N}$. This is the case in most applications; if not, we can utilize an injection of M into \mathbb{N} .

We can decompose M^l according to the cross sum of words: $M^l = \dot{\bigcup}_{b=0}^{\infty} L_b$, where

$$L_b := \{ (x_1, \dots, x_l) \in M^l \mid x_1 + \dots + x_l = b \}.$$

This decomposition is clearly invariant under the action of \mathcal{G} and thus compatible with \sim . For convenience, we will use the short notation $x_\Sigma := x_1 + \dots + x_l$ for the cross sum of $x = (x_1, \dots, x_l)$.

As total order on M^l we use the lexicographical order,

$$(x_1, \dots, x_l) <_{\text{lex}} (y_1, \dots, y_l) \quad :\iff \quad \exists j, 1 \leq j \leq l \text{ such that} \\ x_1 = y_1, x_2 = y_2, \dots, x_{j-1} = y_{j-1} \text{ and } x_j < y_j,$$

and our canonical words are those which are maximal with respect to $<_{\text{lex}}$ among their \sim -class (i.e., their \mathcal{G} -orbit).

Next we define an augmentation operation aug_{word} on M^l . To do so, for $x = (x_1, \dots, x_l) \in M^l$, let $\text{aug}_{\text{word}}(x)$ be the set of words yielding x when the last non-zero entry is decremented: Let $j_0 := \max\{j \mid x_j \neq 0\}$ and put

$$\text{aug}_{\text{word}}(x) := \left\{ \begin{array}{l} (x_1, \dots, x_{j_0} + 1, 0, \dots, 0), \\ (x_1, \dots, x_{j_0}, 1, 0, \dots, 0), \\ (x_1, \dots, x_{j_0}, 0, 1, 0, \dots, 0), \\ \vdots \\ (x_1, \dots, x_{j_0}, 0, 0, \dots, 0, 1) \end{array} \right\}. \quad (4.1)$$

(In the exceptional case $x = (0, \dots, 0)$, assume $j_0 = 1$.)

Theorem 4.3. [Rea78] *aug_{word} is a M_{max}^l -complete augmentation.*

Proof. We have to show that for $b \in \mathbb{N}$,

$$\bigcup_{\substack{x_\Sigma = b \\ x \text{ canonical}}} \text{aug}_{\text{word}}(x) \supseteq \{y \text{ canonical} \mid y_\Sigma = b + 1\}.$$

4. Combinatorial Generation of Translinear Networks

So let $y = (y_1, \dots, y_l)$ be canonical with $y_\Sigma = b + 1$.

Let j be the index determined by $y_{j+1} = \dots = y_l = 0$ and $y_j > 0$ ($j = l$ if $y_l > 0$).

Define $x = (x_1, \dots, x_l)$ by

$$\begin{aligned} x_1 &:= y_1 \\ &\vdots \\ x_{j-1} &:= y_{j-1} \\ x_j &:= y_j - 1 \\ x_{j+1} &:= y_{j+1} = 0 \\ &\vdots \\ x_l &:= y_l = 0 \end{aligned}$$

Obviously, $x_\Sigma = y_\Sigma - 1 = b$ and $y \in \text{aug}_{\text{word}}(x)$. It remains to show that x is canonical, we do this by contradiction: If x is not canonical, there is a $\sigma \in \mathcal{G}$ with $x^\sigma > x$. We show that then also $y^\sigma > y$, which is a contradiction because y is canonical.

Since $x^\sigma > x$, there exists an index k with

$$\begin{aligned} x_{\sigma(1)} &= x_1, \\ &\vdots \\ x_{\sigma(k-1)} &= x_{k-1}, \\ x_{\sigma(k)} &> x_k. \end{aligned}$$

If $k \geq j$, we have

$$\begin{aligned} x_{\sigma(1)} &= x_1, \\ &\vdots \\ x_{\sigma(k-1)} &= x_{k-1}, \\ x_{\sigma(k)} &> x_k, \\ x_{\sigma(k+1)} &\geq 0 = x_{k+1}, \\ &\vdots \\ x_{\sigma(l)} &\geq 0 = x_l, \end{aligned}$$

which means that $(x^\sigma)_\Sigma > x_\Sigma$, obviously a contradiction. So $k < j$. Since $y_\mu \geq x_\mu$ for all $\mu = 1, \dots, l$,

$$\begin{aligned} y_{\sigma(1)} &\geq x_{\sigma(1)} = x_1 = y_1, \\ &\vdots \\ y_{\sigma(k-1)} &\geq x_{\sigma(k-1)} = x_{k-1} = y_{k-1}, \\ y_{\sigma(k)} &\geq x_{\sigma(k)} > x_k = y_k. \end{aligned}$$

hence $y^\sigma > y$, and the proof is finished. □

Having defined canonicity and augmentation, we are now able to catalog the \mathcal{G} -orbits in M^l by successive application of Algorithm 3, starting from $L_0 = \{0\}$. Note that we have the case $\text{aug}(x) \cap \text{aug}(y) = \emptyset$ for $x \neq y$ as mentioned in Remark 4.2.

4.2. Generation of Digraphs

Assume that we wish to build a catalog of the digraphs with n nodes for a given positive integer n . As “digital representation” of digraphs we use adjacency matrices, thus we take

$$L = \mathbb{N}_0^{n \times n}$$

as the set of labeled objects. As the “size” of a digraph, we consider its number of branches, so

$$L_b = \left\{ A \in L \mid \sum_{j,k=1}^n A_{jk} = b \right\}$$

for $b \in \mathbb{N}_0$. (Remember that both parallel branches and self-loops are allowed in our definition of a digraph, see page 7.) This perception of “size” will be quite convenient later on, because for a translinear digraph, it reflects the number of transistors of the respective network.

If we view an $n \times n$ -matrix as a word of length n^2 , consisting of the concatenation of the matrix rows, the number of branches is just the cross sum. That means, we can apply the generation of words presented in Subsection 4.1.3, to generate adjacency matrices.

The permutation group to consider on the words of length n^2 is the image of the group monomorphism

$$\iota : S_n \hookrightarrow S_{n^2}$$

defined by

$$\iota(\sigma)(nj + k) = \sigma(j)n + \sigma(k)$$

for $\sigma \in S_n$, $j = 0, \dots, n-1$ and $k = 1, \dots, n$. It will be more convenient, however, to regard the action of S_n on the set of matrices, instead of a subgroup of S_{n^2} acting on the set of words.

An algorithm for the generation of all digraphs with n nodes is derived easily as a specialisation of Algorithm 3:

Algorithm 4 generateDigraphs(integer $n > 0$)

```

1:  $U_0 := \{\mathbf{0}_n\}$ 
2: for  $b := 1$  to  $\infty$  do
3:    $U_b := \emptyset$ 
4:   for all  $A \in U_{b-1}$  do
5:     for all  $A' \in \text{aug}_{\text{word}}(A)$  do
6:       if ISCANONICALDIGRAPH ( $A'$ ) then
7:          $U_b := U_b \cup \{A'\}$ 
8:       end if
9:     end for
10:  end for
11:  Output:  $U_b$ 
12: end for

```

(Here, $\mathbf{0}_n$ denotes the $n \times n$ zero matrix.)

We can explicitly write down the canonicity test needed in line 6 of Algorithm 4:

Algorithm 5 isCanocicalDigraph(square matrix A)

```

1:  $n := n(A)$ 
2: for all  $\sigma \in S_n$  do
3:   if  $P_\sigma^{-1}AP_\sigma >_{\text{lex}} A$  then
4:     return false
5:   end if
6: end for
7: return true

```

The $>_{\text{lex}}$ comparison in line 3 of Algorithm 5 is meant as lexicographical comparison of the words formed by concatenation of the matrix rows, and $n(A)$ in line 1 means the number of rows (which is equal to the number of columns) of A .

Algorithm 4 is the basis for sophisticated and very fast graph generation systems that are used in particular for listing chemical isomers [Rea78, Ker99, Hea72, Gru93, GLM97, PR87, Pól37].

Algorithms 4 and 5 give a method to generate digraphs with a given number of nodes.

When we use a digraph to model an electrical network, the number of branches is closely linked with the number of network elements. Thus it is indeed an appropriate measure of “size” or “complexity” of the network. The number of nodes, in contrast, is quite irrelevant. Therefore it is quite inconvenient for us that Algorithm 4 needs a number of nodes to be prescribed. We would prefer an algorithm which generates digraphs independly of the number of nodes.

In the following, such an algorithm is developed.

4. Combinatorial Generation of Translinear Networks

Up to now, we use an augmentation that adds exactly one branch to a digraph and leaves the node set fixed. To get away from the fixed number of nodes, we can allow the insertion of new nodes while adding a branch: We consider not only the insertion of a branch between two existing nodes, but also a new branch from an existing to a new node or vice versa, or even a new isolated branch between two new nodes. We restrict to digraphs without isolated nodes. (It would be trivial, however, to generate all digraphs from those without isolated nodes. For our purpose, however, disconnected digraphs are irrelevant.)

As before it is very helpful to discuss the augmentation by means of its reversal, the deletion of branches. We distinguish 4 cases when we delete a branch:

1. Neither tail nor head of the branch becomes isolated. This is the “standard” case which Algorithm 3 restricts to.
2. The head node becomes isolated and vanishes. The tail node stays.
3. The tail node becomes isolated and vanishes. The head node stays.
4. Both head node and tail node become isolated and vanish.

The dimension (number of rows/columns) of the corresponding adjacency matrix of shrinks by 1 in cases 2 and 3, and it shrinks by 2 in case 4. In case 1, of course, the dimension does not change.

We now consider in detail, for the 4 cases, how the adjacency matrix of the augmented digraphs can look like if the adjacency matrix of the branch-deleted digraph is A .

1. The branch is inserted between two existing nodes. The adjacency matrix of the augmented digraph is one from $\text{aug}_{\text{word}}(A)$.
2. The new branch points from an existing node to a new node. The adjacency matrix of the augmented digraph is one of

$$A^{\rightarrow 1} := \begin{bmatrix} & & & 1 \\ & & & 0 \\ & A & & \vdots \\ & & & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix}, \dots, A^{\rightarrow n} := \begin{bmatrix} & & & 0 \\ & & & \vdots \\ & A & & 0 \\ & & & 1 \\ 0 & \dots & 0 & 0 \end{bmatrix}.$$

3. The new branch points from a new node to an existing node. The adjacency matrix of the augmented digraph is one of

$$A^{\leftarrow 1} := \begin{bmatrix} & & & & 0 \\ & & & & \vdots \\ & A & & & 0 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix}, \dots, A^{\leftarrow n} := \begin{bmatrix} & & & & 0 \\ & & & & \vdots \\ & A & & & 0 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix}.$$

4. Combinatorial Generation of Translinear Networks

4. The new branch points from one new node to another. The adjacency matrix of the augmented digraph is

$$A^{\leftrightarrow} := \begin{bmatrix} & & & 0 & 0 \\ & A & & \vdots & \vdots \\ & & & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

(We do not take into account the matrix

$$\begin{bmatrix} & & & 0 & 0 \\ & A & & \vdots & \vdots \\ & & & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix},$$

because it is not canonical.)

Remark 4.4. If A is canonical, augmentations according to 2 or 4 always yield canonical matrices. This is not the case for augmentations according to 3.

We can use the following (non-terminating) algorithm to generate literally all digraphs without isolated nodes and without self-loops.

Algorithm 6 crossGenerateDigraph

```

1:  $U_0 := \{(0)\}$ 
2: for  $b := 1$  to  $\infty$  do
3:    $U_b := \emptyset$ 
4:   for all  $A \in U_{b-1}$  do
5:     for all  $A' \in \text{aug}_{\text{word}}(A)$  do
6:       if ISCANONICALDIGRAPH( $A'$ ) then
7:          $U_b := U_b \cup \{A'\}$ 
8:       end if
9:     end for
10:    for  $j = 1$  to  $n(A)$  do
11:       $U_b := U_b \cup \{A^{\rightarrow j}\}$ 
12:    end for
13:    for  $j = 1$  to  $n(A)$  do
14:      if ISCANONICALDIGRAPH( $A^{\leftarrow j}$ ) then
15:         $U_b := U_b \cup \{A^{\leftarrow j}\}$ 
16:      end if
17:    end for
18:     $U_b := U_b \cup \{A^{\leftrightarrow}\}$ 
19:  end for
20:  Output:  $U_b$ 
21: end for

```

4.3. Generation of Translinear Digraphs

In this section, we explain how to adapt Algorithm 6 for the generation of translinear digraphs.

First, in Subsection 4.3.1, we investigate the special structure of adjacency matrices of layered digraphs. Subsection 4.3.3 presents the augmentation and Subsection 4.3.2 the canonicity we use for layered digraphs. Subsection 4.3.4 briefly tells how to restrict to translinear digraphs when applying the generation of layered digraphs.

We can dispense with pseudocode in this section, the structure of the algorithms is clear from the pseudocode in the preceding sections.

4.3.1. Adjacency matrices of layer digraphs

Consider the adjacency matrix of a layered digraph. Assume the digraph is connected, so the rank of each node is well-defined. Because we know that there can never be a branch from node v to node w if $\text{rank}(w) \neq \text{rank}(v) + 1$, we only need to store adjacency matrices “from one layer to the next”. If the maximal rank of any node of G is R , and the number of nodes of rank k is n_k , we need R matrices $A^{(1)}, \dots, A^{(R)}$, where $A^{(k)}$ has

4. Combinatorial Generation of Translinear Networks

dimensions $n_{k-1} \times n_k$. This makes a total of $\sum_{k=1}^R n_{k-1}n_k$ entries, which is always less than the $n(n-1)/2$ entries we need to store for the complete adjacency matrix.³ (We continue to denote the total number of nodes by n , so here $n = n_0 + n_1 + \dots + n_R$.) The matrices $A^{(1)}, \dots, A^{(R)}$ can be regarded as blocks of the complete adjacency matrix, if we order the nodes according to their rank. Outside of these blocks are then only zeros:

	rank 0	rank 1	rank 2		rank $R-1$	rank R
rank 0	0	0	0		0	0
rank 1	$A^{(1)}$	0	0		0	0
rank 2	0	$A^{(2)}$	0			
			\ddots	\ddots		
rank $R-1$	0	0	0	\ddots	0	0
rank R	0	0	0		$A^{(R)}$	0

So we represent a layered digraph by a tuple of matrices. Our set of labeled structures is

$$L = \left\{ (A^{(1)}, \dots, A^{(R)}) \left| \begin{array}{l} R \text{ positive integer,} \\ A^{(k)} \in \{0, 1\}^{n_{k-1} \times n_k} \text{ for } k = 1, \dots, R, \\ n_0, \dots, n_R \text{ positive integers} \end{array} \right. \right\}.$$

The “size” is still the total number of branches, namely

$$b = \sum_{k=1}^R \sum_{i=1}^{n_{k-1}} \sum_{j=1}^{n_k} A_{ij}^{(k)}.$$

Isomorphisms of layered digraphs consist of bijections of the node sets that preserve rank. These bijections decompose into permutations “inside” each rank. The equivalence relation to consider is the following.

Definition 4.5. For $A = (A^{(1)}, \dots, A^{(R_1)})$, $B = (B^{(1)}, \dots, B^{(R_2)}) \in L$, we define $A \sim B$ if and only if A and B have the same dimensions (n_0, \dots, n_R) (in particular $R_1 = R_2 = R$) and there exist permutations $\sigma_0 \in S_{n_0}, \dots, \sigma_R \in S_{n_R}$ such that for $k = 1, \dots, R$, $i = 1, \dots, n_{k-1}$, $j = 1, \dots, n_k$:

$$B_{ij}^{(k)} = A_{\sigma_{k-1}(i)\sigma_k(j)}^{(k)}.$$

4.3.2. Canonicity

We view a matrix tuple $(A^{(1)}, \dots, A^{(R)})$ as word of length $\sum_{k=1}^R n_{k-1}n_k$ by reading the matrices one after the other, and each matrix word by word. Thus, a matrix tuple

³The number $n(n-1)/2$ takes into account that a translinear digraph is acyclic, so with an appropriate numbering of the nodes, the adjacency matrix has zeros in the lower triangular part as well as on the diagonal.

$(A^{(1)}, \dots, A^{(R)})$ is canonical if and only if for all $(\sigma_0, \dots, \sigma_R) \in S_{n_0} \times \dots \times S_{n_R}$, there is an integer j , $1 \leq j \leq R$, such that

$$\begin{aligned} A^{(1)} &= P_{\sigma_1}^{-1} A^{(1)} P_{\sigma_0}, \\ &\vdots \\ A^{(j)} &= P_{\sigma_{j-1}}^{-1} A^{(j)} P_{\sigma_{j-2}}, \\ A^{(j+1)} &>_{\text{lex}} P_{\sigma_j}^{-1} A^{(j+1)} P_{\sigma_{j-1}}, \end{aligned}$$

where the inequation is to be omitted if $j = R$.

The canonicity test can be performed by testing all permutation tuples $(\sigma_0, \dots, \sigma_R) \in S_{n_0} \times \dots \times S_{n_R}$.

(For the implementation of the class “ranked” (see Appendix B), a refined and more efficient version of the canonicity test is used that incorporates the successive computation of the automorphism group of the layered digraph.)

4.3.3. Augmentation

As we distinguished 4 “augmentation cases” in the general digraph case, we distinguish 6 augmentation cases of layered digraphs. We classify them as sub-cases of the general digraph cases as follows:

1. No new node appears. The new branch is inserted between layer R and layer $R-1$. A_R is replaced by an element of $\text{aug}_{\text{word}}(A_R)$.
2. The new branch points from existing node of rank R to a new node of rank $R-1$. A_R is replaced by

$$\begin{bmatrix} & 0 \\ & \vdots \\ A_R & 0 \\ & 1 \end{bmatrix}, \quad (4.2)$$

and a zero row is appended to A_{R-1} . (In contrast to the general digraph case, there is only one possibility of this augmentation. The “1” does not appear at another than the last position of the new column in eqn. (4.2), because otherwise decrementing the last non-zero entry would not yield A_R .)

3. The new branch points from a new node to an existing node.
 - a) The new node appears in the top layer, the new branch points from layer R to layer $R-1$. A_R is replaced by one of

$$\begin{bmatrix} & A_R \\ 1 & 0 & \dots & 0 \end{bmatrix}, \dots, \begin{bmatrix} & A_R \\ 0 & \dots & 0 & 1 \end{bmatrix}.$$

4. Combinatorial Generation of Translinear Networks

- b) The new node appears “above” the old top layer. A matrix A_{R+1} is appended to the matrix tuple, where A_{R+1} is one of

$$[1 \ 0 \ \cdots \ 0], \dots, [0 \ \cdots \ 0 \ 1].$$

4. Both head and tail of the inserted branch are new nodes.

- a) The new tail node has rank R , the new head node has rank $R - 1$. A_R is replaced by

$$\begin{bmatrix} & & & 0 \\ & A_R & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix},$$

and a zero row is appended to A_{R-1} .

- b) The new tail node has rank $R+1$ (in particular, a new top layer is introduced), the new head node has rank R . A zero row is appended to A_R , and the matrix

$$A_{R+1} = [0 \ \cdots \ 0 \ 1]$$

is appended to the matrix tuple. Note that the represented digraph is disconnected and that all further *canonical* augmentations of the matrix tuple will represent a disconnected digraph.

4.3.4. Specialization for Translinear Digraphs

Having derived techniques for the generation of layered digraph, we specialise them for the generation of translinear digraphs. As translinear digraphs are nothing but biconnected layered digraphs, we can simply generate all layered digraphs and filter them for the biconnected ones. The algorithm of Tarjan [Jun99, p. 343] is available for efficiently testing a (di)graph for biconnectedness.

We can increase the efficiency of the generation of translinear digraphs if we do not generate *all* layered digraphs but try to leave out those which are for sure never augmented to a biconnected one (that is, not even after several augmentations). The first measure we take is to omit the augmentation 4b, because resulting digraphs would be disconnected. As second measure, we do not apply augmentation 3b whenever there are any leaf nodes (nodes with only one branch incident) in layers $0, \dots, R - 1$. Besides 4b, augmentation 3b is the only augmentation introducing a new layer, and obviously, leaf nodes in the lower layers would stay “forever”, so that none of the digraphs obtained by repeated augmentation would be biconnected.

Orderly generation of translinear digraphs as described has been implemented in C++ and applied to list all translinear digraphs with 9 or less branches. As results, Figure 4.2

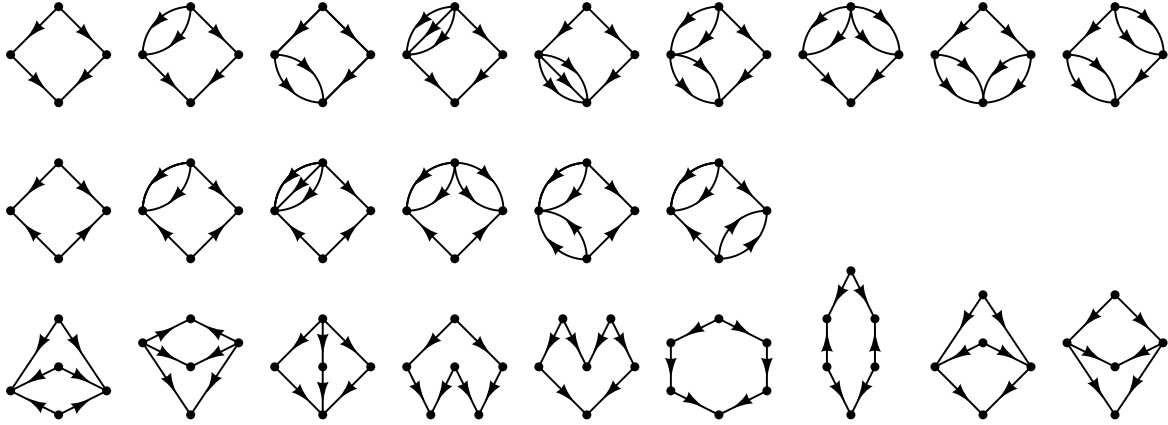


Figure 4.2.: The translinear digraphs with 6 or less branches.

shows the translinear digraphs with up to 6 branches⁴, and the following table shows the numbers of the translinear digraphs with $b = 4, \dots, 9$ branches.

number of branches:	4	5	6	7	8	9
number of translinear digraphs:	2	3	19	39	174	559

4.4. Generation of Collector Assignments

In this section, we consider the generation of all collector assignments for a given translinear digraph G .

Definition 4.6. Two collector assignments C_1, C_2 on G are called **isomorphic** if there is a digraph automorphism $\varphi : G \rightarrow G$ such that $C_1(\varphi(e)) = \varphi(C_2(e))$ for every branch $e \in E(G)$, where $\varphi(v_{\text{out}+}) := v_{\text{out}+}$, $\varphi(v_{\text{out}-}) := v_{\text{out}-}$ and $\varphi(v_{\text{void}}) := v_{\text{void}}$.

Definition 4.6 reflects topological equivalence of collector assignments. However, we decide to use a broader notion of equivalent collector assignments, where the isomorphism classes comprise objects that might represent slightly different topologies, but whose system of network equations will look the same.

Definition 4.7. Two collector assignments C_1, C_2 on G are called **generalized isomorphic** if there exist a digraph automorphism $\varphi : G \rightarrow G$ and a permutation σ on $V(G)$ such that σ permutes only the top layer of G and

$$\varphi(C_2(e)) = C_1(\varphi(e))$$

⁴Admittedly, the list of the 24 translinear digraphs with up to 6 branches can as well be compiled without computer. For space reasons, we do not list translinear digraphs with more branches, since such a listing could only consist of a computer-generated display of the adjacency matrices and not of the more pleasant diagrams.

whenever $\text{rank}(C_2(e)) < R$ and

$$\sigma(C_2(e)) = C_1(\varphi(e))$$

whenever $\text{rank}(C_2(e)) = R$.

A permutation of the top layer nodes doesn't affect the network equations, because the corresponding node equations are just permuted among themselves as well, but not changed. The reason for this, in turn, is that there cannot be any emitters connected to a top layer node. For nodes where emitters are connected to, the emitter contributions to the node equations would “stay”, whereas the collector contributions “follow” the permutation. (Here, the irrelevance of base currents is crucial. See Subsection 5.1.2 for a detailed formulation of the node equations.)

To represent a collector assignment C digitally, we fix an order e_1, \dots, e_b of the branches of G and use a tuple of integers $X(C) = (c_1, \dots, c_b)$, where c_j identifies the node $C(e_j)$. On the set of these b -tuples of integers, we consider the induced action of the automorphism group $\text{Aut}(G)$. Since an automorphism φ of G permutes the branches as well as the nodes, it not only permutes the *positions* of the integers in $X(C)$, it also changes the integers themselves.

Unfortunately, this means that we cannot reduce the cataloging problem for these integer tuples to the cataloging problem of words as it is formulated in Subsection 4.1.3, because the latter is restricted to the case where isomorphic words differ only by a pure permutation of the positions.

But still, we can define a canonical integer tuple to be the lexicographically maximal one among its $\text{Aut}(G)$ -orbit. Taking into account the generalized isomorphism of collector assignments according to Definition 4.7, we adjust:

$X(C)$ is canonical if it is maximal in its $\text{Aut}(G) \times S_{n_R}$ -orbit. Here, S_{n_R} is considered as the symmetric group on V_R , the set of nodes of rank R . (A permutation of these top layer nodes changes the corresponding integers c_j , but not their positions.)

We have seen that canonicity can be applied, but augmentation can not. Because of the latter, we fall back to Algorithm 2 to generate the collector assignments on a given translinear digraph.

4.5. Cataloging Formal Networks

We have derived algorithms for the generation of translinear digraphs and for the generation of collector assignments for a particular translinear digraph. Given a pair (G, C) of a digraph and a collector assignment, it is trivial to derive all possible complete formal network descriptions in terms of Definition 3.17: Just list the triples (G, C, v_0) , where $v_0 \in V(G) \setminus \text{img}(C)$.

4. Combinatorial Generation of Translinear Networks

For various reasons it may happen that the circuit modelled by a formal network (G, C, v_0) is of no practical value. For example, consider a *source* v of G , that is, a node which is not the head of any branch. No emitter is connected to v . Since we assume an independent input current applied to v unless $v = v_0$, we get into trouble if no collector is connected to v as well, because the bases cannot compensate the externally forced current. Consequently, we discard formal networks possessing a source v of G with $C(e) \neq v$ for all branches e .

Obviously, a network without any output is of no use. Therefore, only collector assignments with $C(e) \in \{v_{\text{out}+}, v_{\text{out}-}\}$ for at least one branch e are to be considered. Furthermore, if there is no branch e with $C(e) = v_{\text{out}+}$, the output of the network is

$$y = - \sum_{C(e)=v_{\text{out}-}} x_e$$

(compare eqn. (3.21)). We assume that for applications an inverted output $-y$ is equally of benefit as y is, so we can swap $v_{\text{out}+}$ and $v_{\text{out}-}$. Thus we can require $C(e) = v_{\text{out}+}$ for at least one branch e without losing any non-redundant formal network.

Both of the stated “practical value” conditions are incorporated into the following definitions.

Definition 4.8. We call a formal translinear network (G, C, v_0) **valid**, if $C(e) = v_{\text{out}+}$ for at least one $e \in E(G)$, and for each source v of G , either $v = v_0$ or there is at least one $e \in E(G)$ with $C(e) = v$.

A collector assignment is valid, if we can be sure that it leads to a valid formal network:

Definition 4.9. We call a collector assignment C on a translinear digraph G **valid**, if $C(e) = v_{\text{out}+}$ for at least one $e \in E(G)$ and there is at most one source v of G such that $v \notin \text{img}(C)$.

(In case there is exactly one such source node, it can and must be chosen as ground node.)

The combinatorial methods presented in this chapter have been implemented in C++ and have been used to generate exhaustive and non-redundant lists of formal translinear networks with up to 8 transistors. The following table shows the numbers of pairs (G, C) of a translinear digraph G and a collector assignment C on G , as generated by the program `tlgen` (see Appendix B), and the numbers of those pairs where C is valid.

number of transistors	pairs (G, C)	valid pairs
4	250	177
5	2248	1868
6	51930	42102
7	1149476	978812
8	32125843	27335135

4. *Combinatorial Generation of Translinear Networks*

For up to 6 transistors, Appendix A shows how the numbers in the right column split up for the particular digraphs.

5. A Catalog of Topologies as a Synthesis Tool

Thanks to the combinatorial algorithms presented in Chapter 4, a catalog of the static formal translinear networks with up to 8 transistors is available. The formal networks are given as tripels (G, C, v_0) as specified by Definition 3.17. In this chapter, we show how the catalog can effectively put into use as a resource for network synthesis.

Throughout the chapter, we will denote the input nodes of a formal network $N = (G, C, v_0)$ by v_1, \dots, v_{n-1} (thus $V(G) = \{v_0, \dots, v_{n-1}\}$) and the branches of G by e_1, \dots, e_b .

5.1. Translinear Network Equations

In this section, we investigate how the system of network equations looks like for a formal translinear network.

We remind that in general, the behavior of an electrical network is described by

- the loop equations (given by Kirchhoff's Voltage Law),
- the node equations (given by Kirchhoff's Current Law),
- and the element equations.

In the case of a translinear network, the loop equations and the element equations (namely the exponential transistor model) are merged by the translinear principle and result in the translinear loop equations.

Thus, the behavior of a static translinear network is entirely described by the translinear loop equations and the node equations. For dynamic translinear networks, one has to add the equations resulting from the dynamic translinear principle (eqn. (3.8)) to make the behavioral network description complete.

5.1.1. Translinear Loop Equations

Consider the translinear digraph G of a formal network. We have seen in Section 2.2 that the ideal generated by $\{B_S | S \text{ loop of } G\} \subset \mathbb{Q}[x_1, \dots, x_b]$ is exactly the toric ideal I_G of G . If we interpret each variable x_j as the collector current of the transistor corresponding to branch e_j of G , the translinear loop equation for a loop S is $B_S = 0$. That means that we can use I_G to conveniently capture all of the network's translinear loop equations.

Remark 5.1. Since G is a layered digraph (see Definition 3.8), I_G is homogeneous. This is quite easy to see, because for a loop S of any digraph G , the polynomial B_S is homogenous if and only if S has as many forward branches as backward branches, so I_G is homogeneous if and only if G is layered.

From Lemma 2.33 we know how we can compute the toric ideal of a digraph: Take a fundamental loop system S_1, \dots, S_c of G , then I_G is the saturation of $\langle B_{S_1}, \dots, B_{S_c} \rangle$.

5.1.2. Node Equations

Consider an input node $v_j \in V(G)$, $1 \leq j \leq n-1$, of a formal translinear network $N = (G, C, v_0)$. Since we neglect the base currents of the transistors, the only contributions to the node equation of v_j come from emitters, collectors, and the independent input current i_j (in Subsection 3.4.2 denoted by i_{v_j}). Thus, the node equation of v_j is

$$0 = i_j + \sum_{\substack{1 \leq k \leq b \\ \text{head}(x_k) = v_j}} x_k - \sum_{\substack{1 \leq k \leq b \\ C(x_k) = v_j}} x_k.$$

5.1.3. The Network Ideal

For a formal translinear network $N = (G, C, v_0)$, we consider the system consisting of

- the translinear loop equations,
- the node equations,
- and the output equation $y = \sum_{C(e_k) = v_{\text{out}+}} x_k - \sum_{C(e_k) = v_{\text{out}-}} x_k$ (compare eqn. (3.21)).

We call the ideal generated by the corresponding polynomials the *network ideal*:

Definition 5.2. Let $N = (G, C, v_0)$ be a formal static translinear network. The **network ideal** $I_N \subset \mathbb{Q}[x_1, \dots, x_b, i_1, \dots, i_{n-1}, y]$ is the ideal generated by

- the toric ideal $I_G \subset \mathbb{Q}[x_1, \dots, x_{n-1}] \subset \mathbb{Q}[x_1, \dots, x_b, i_1, \dots, i_{n-1}, y]$,

- the polynomials

$$-i_j + \sum_{\text{head}(e_k)=v_j} x_k - \sum_{C(e_k)=v_j} x_k \quad (5.1)$$

for $j = 1, \dots, n - 1$,

- and the polynomial

$$-y + \sum_{C(e_k)=v_{\text{out}+}} x_{e_k} - \sum_{C(e_k)=v_{\text{out}-}} x_{e_k} .$$

Remark 5.3. I_N is homogeneous and prime. This can easily be deduced from the fact that I_G is homogeneous and prime (see Remarks 2.24 and 5.1) and the fact that each remaining generator is linear and contains a variable not contained by any other generator.

5.1.4. Elimination of Collector Currents

As inputs of a formal translinear network we have the input currents i_1, \dots, i_{n-1} , the output is y . To get a direct input-output relationship, we eliminate the remaining variables, the collector currents x_1, \dots, x_b , from the network ideal. We denote the resulting ideal by I'_N :

$$I'_N := I_N \cap \mathbb{Q}[i_1, \dots, i_{n-1}, y].$$

The computer algebra system SINGULAR [GPS01] has been used to compute I_N and to perform the elimination for each formal network in the catalog with 7 or less transistors. It has been found that every time the inputs i_1, \dots, i_{n-1} , expressed as linear combinations of the collector currents via eqn. (5.1), are linearly independent, I'_N turns out to be a principle ideal. In other words, in these cases the elimination of collector currents always yielded a single polynomial equation $f_N(i_1, \dots, i_{n-1}, y) = 0$ in the input and output currents.

It is very probably a general fact that I'_N is a principle ideal whenever i_1, \dots, i_{n-1} are linearly independent. Computations for some networks with more than 7 transistors support this, too. A proof, however, has not been found.

The catalog of formal networks has been equipped with a generator $f_N \in \mathbb{Z}[i_1, \dots, i_{n-1}, y]$ of I'_N for every formal network N .

Remark 5.4. Since I_N is homogeneous and prime, so are I'_N and f_N .

Remark 5.5. Depending on which collectors are assigned to $v_{\text{out}+}$ and $v_{\text{out}-}$, it may happen that the translinear loop equations are simply thrown away during the elimination. This is the case if and only if f_N is linear and represents a kind of degeneration of the network N : The input-output relation is not at all influenced by the translinear loops! We can ignore these degenerated networks, the linear behavior can as well be achieved by pure current addition/substraction via node equations, without any transistors.

5.2. The Input Matrix

Assume a circuit is to be designed whose desired behaviour is given by a polynomial equation $g(u_1, \dots, u_s, y) = 0$, where u_1, \dots, u_s are input variables and y is an output variable. (For simplicity we restrict to the case of only one output.) We assume that g has integer coefficients and is irreducible.

We now want to test all formal translinear networks from our catalog whether they can be used to implement g . Since we have computed for each formal network $N = (G, C, v_0)$ in the catalog a polynomial $f_N \in \mathbb{Z}[i_1, \dots, i_{n-1}, y]$ describing its behavior, we test the suitability of a particular formal network N by testing whether f_N “matches” g .

This matching test is not as simple as it appears: We must clarify how to identify the inputs u_1, \dots, u_s with the inputs currents i_1, \dots, i_{n-1} of the formal network.

We assume that for each input u_k , we can use as many current sources as we want, each delivering the same current u_k . In our model, each of these current sources is connected between the ground node v_0 and one of the input nodes v_1, \dots, v_{n-1} . Both orientations are possible, so that the current source contributes either u_k or $-u_k$ to the respective input current i_j . Of course, several sources can be connected to one input node, even several sources with the same input current u_k . Thus, the relation of the available inputs u_1, \dots, u_s with the formal node inputs i_1, \dots, i_{n-1} is

$$i_j = \sum_{k=1}^s b_{jk} u_k$$

for $j = 1, \dots, n-1$ with $b_{jk} \in \mathbb{Z}$. Without additional circuitry, an input node cannot be supplied with anything else than a finite sum of the currents delivered by the available sources.

We call the matrix $B := (b_{jk}) \in \mathbb{Z}^{(n-1) \times s}$ the **input matrix**

Example 5.6. For the network in Figure 5.1,

$$\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} = \begin{pmatrix} -1 & -1 \\ 0 & -1 \\ 0 & 2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$

so $B = \begin{pmatrix} -1 & -1 \\ 0 & -1 \\ 0 & 2 \\ 1 & 0 \end{pmatrix}$.

To make the polynomials f_N and g “comparable”, we map f_N to $\mathbb{Z}[u_1, \dots, u_s, y]$ via

$$\begin{aligned} \varphi_B : \mathbb{Z}[i_1, \dots, i_{n-1}, y] &\rightarrow \mathbb{Z}[u_1, \dots, u_s, y], \\ i_1 &\mapsto b_{1,1}u_1 + \dots + b_{1,s}u_s, \\ &\vdots \\ i_{n-1} &\mapsto b_{n-1,1}u_1 + \dots + b_{n-1,s}u_s, \\ y &\mapsto y. \end{aligned}$$

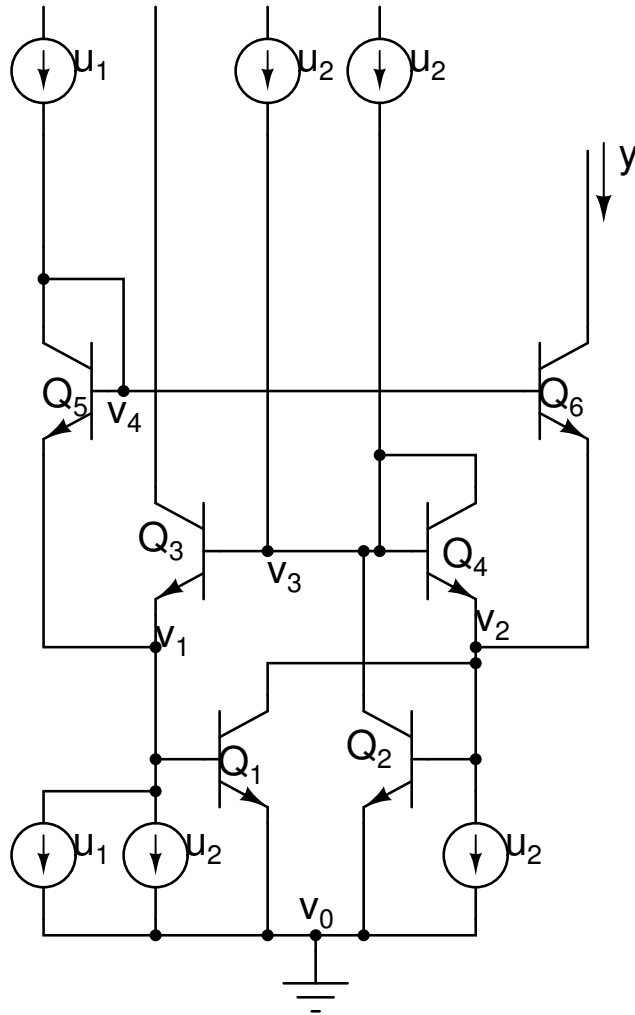


Figure 5.1.: A translinear network.

We can now formulate precisely what we mean by a match of f_N and g :

Definition 5.7. Let $f \in \mathbb{Z}[i_1, \dots, i_{n-1}, y]$ and $g \in \mathbb{Z}[u_1, \dots, u_s, y]$. A **match of f onto g** is a matrix $B \in \mathbb{Z}^{(n-1) \times s}$ such that $g = \lambda \varphi_B(f_N)$ for some $\lambda \in \mathbb{Q}^*$.

(Obviously, we allow multiplication by a nonzero constant λ because $g = 0 \Leftrightarrow \lambda g = 0$.)

For practical reasons, we restrict to matches where the coefficients b_{jk} are bounded by an integer l , usually $l = 1$ or $l = 2$. By the *matching problem*, we mean the problem to determine all matches $B \in \{-l, -l + 1, \dots, l - 1, l\}^{(n-1) \times s}$ of f_N onto g .

5.3. Solution of the Matching Problem

If the total degree of g does not equal the total degree of f_N , we know that no match of f_N onto g exists. From now on we assume equality of the degrees and define $\deg g = \deg f_N =: d$.

We solve the matching problem algorithmically by comparison of coefficients: $g = \lambda \varphi_B(f_N)$ if and only if every coefficient of the polynomial $g - \lambda \varphi_B(f_N)$ vanishes. Regarding the coefficients as polynomials in the entries of B and in λ , we get a system of polynomial equations which must be solved by the entries of every B that is a match.

Formally, we introduce indeterminates $\mathbf{w} := \{w_{jk}\}_{\substack{j=1, \dots, n-1 \\ k=1, \dots, s}}$ for the entries b_{jk} of B and Λ for λ and consider the homomorphism of rings

$$\begin{aligned} \varphi : \mathbb{Q}[i_1, \dots, i_{n-1}, y] &\rightarrow \mathbb{Q}[u_1, \dots, u_s, y, \mathbf{w}, \Lambda] \\ i_j &\mapsto \sum_{k=1}^s w_{jk} u_k. \end{aligned}$$

The ideal

$$I_{\text{coef}} := \langle \text{coef}(\Lambda \varphi(f_N) - g, M) \mid M \in \text{Mon}_d(u_1, \dots, u_s, y) \rangle \subset \mathbb{Q}[\mathbf{w}, \Lambda]$$

(where $\text{Mon}_d(u_1, \dots, u_s, y)$ is the set of monomials of degree d in u_1, \dots, u_s, y) represents the system of equations which the entries of B have to satisfy to be a “match”.

To force the entries of B to be integers in the range $\{-l, \dots, l\}$, we demand additionally

$$w_{jk} \cdot (w_{jk} - 1)(w_{jk} + 1) \cdots (w_{jk} - l)(w_{jk} + l) = 0$$

for $j = 1, \dots, n - 1$ and $k = 1, \dots, s$.

We need to compute the zero set $Z(I_0)$ of the ideal

$$I_0 := \left\langle I_{\text{coef}}, \left\{ \prod_{\nu=-l}^l (w_{jk} - \nu) \right\}_{j,k} \right\rangle \subset \mathbb{Q}[\mathbf{w}, \Lambda].$$

By our artificial inclusion of $\left\{ \prod_{\nu=-l}^l (w_{jk} - \nu) \right\}_{j,k}$, the ideal is zero-dimensional and there are no irrational points.

We can compute these points of $Z(I_0)$ using Gröbner Basis methods. For example, a computation of the minimal associated primes of I_0 reveals maximal ideals representing the points of $Z(I_0)$ and thus the matches we are looking for.

To increase efficiency, the computation can be performed over a finite field \mathbb{Z}/p instead of \mathbb{Q} . In this case, it is possible that solutions over \mathbb{Z}/p appear which are no solutions over \mathbb{Q} . This can be avoided by choosing the prime number p sufficiently large.

A SINGULAR procedure has been written that determines all matches of a given network polynomial f_N onto a given “target behavior” polynomial g , see Appendix B. As further parameters, the procedure takes the bound l and a prime number p , so that the user can specify over which finite field the computation shall be performed.

5.4. Final Network Check

Assume we have found a match, so that we know a formal network $N = (G, C, v_0)$ and an input matrix B such that $\varphi_B(f_N) = 0$ is exactly the behavior we want.

In the system of network equations (see Subsection 5.1.1), we can replace i_j by $\sum_{k=1}^s b_{jk} u_k$ for $j = 1, \dots, n-1$ and solve it for the collector currents x_1, \dots, x_b , obtaining them as functions of u_1, \dots, u_s .

There is a finite number of solutions $x_\nu^{(1)}(u_1, \dots, u_s), \dots, x_\nu^{(d_\nu)}(u_1, \dots, u_s)$ for each collector current.

5.4.1. Positivity Check of Collector Currents

All collector currents must be positive, otherwise our transistor model (eqn. (3.1)) is invalid.

Usually, the specification of the desired network behavior includes a range $u_k^{\min} < u_k < u_k^{\max}$ for each input, $u_k^{\min} \in \mathbb{R} \cup \{-\infty\}$, $u_k^{\max} \in \mathbb{R} \cup \{\infty\}$. Given these ranges, we can check for each collector current x_ν which of the solutions $x_\nu^{(\xi)}$ violate the condition $x_\nu^{(\xi)}(u_1, \dots, u_s) > 0$.

The extended functionality (“simplification using assumptions”) of the MATHEMATICA function `Simplify` [Wol99] performs this check satisfactorily.

In all example computations it was found that there is at most one $\xi \in \{1, \dots, d_\nu\}$ with $x_\nu^{(\xi)}(\mathbf{u}) > 0$ for all $\mathbf{u} \in]u_1^{\min}, u_1^{\max}[\times \dots \times]u_s^{\min}, u_s^{\max}[$. If there is none, a hardware implementation of the network will not work, because the model assumption of positive collector currents is not satisfied. We can delete the pair (N, B) from our set of “candidates”.

But if there is indeed a positive solution for each collector current, we have determined the exact explicit dependence of the collector currents on u_1, \dots, u_s , and thus also the exact explicit dependence of the output

$$y = \sum_{C(e)=v_{\text{out}+}} x_e - \sum_{C(e)=v_{\text{out}-}} x_e$$

on u_1, \dots, u_s .

5.4.2. Output Function Check

After the positivity check of collector currents, we have an explicit description of the network behavior in the form $y = h(u_1, \dots, u_s)$, and we know that

$$g(u_1, \dots, u_s, h(u_1, \dots, u_s)) = 0.$$

Still, it might be that the intended behavior was a different branch of the solution of the polynomial equation $g = 0$. (For example, we may look for a network with $y = \sqrt{u_1}$ and thus specify $g = y^2 - u_1$. Then the network search may yield a network with $y = -\sqrt{u_1}$. Of course, less pathological examples exist where the difference is more than the sign.)

The comparison of h with the intended explicit behavior is our last check to filter out formal networks which are of no practical relevance.

Both the positivity check of collector currents and the output function check have been implemented in MATHEMATICA.

6. Example Application

The following example comes from industrial applications at the company Analog Microelectronics GmbH (in the following: AMG). The problems, communicated by E. J. Roebbers from AMG, concern the question whether the nonlinear behavior of a certain sensor device can be compensated by a static translinear circuit. This chapter is joint work with E. J. Roebbers.

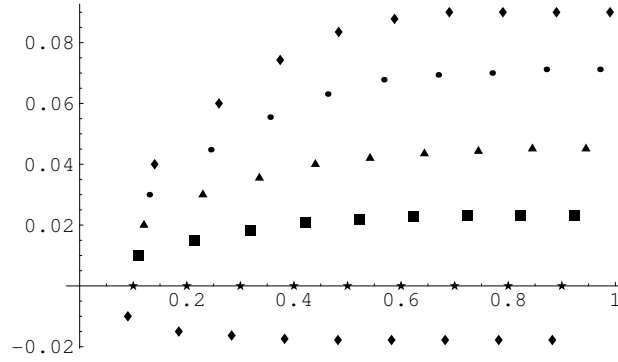
A humidity sensor subsystem for air conditioning is in development at AMG. The sensor device to be used is optimized for an operating temperature of 23°C; at this temperature, its output is virtually proportional to the relative humidity. For other temperatures, it shows deviations from this linear behavior which are to be compensated. For this purpose, an “analog computation” circuit is desired to reconstruct the deviation. The circuit of course needs a second input where information about the actual temperature is provided independently of the sensor output.

We denote the real relative humidity by H , the output of the sensor (the “measured humidity”) by H_m . The following information about the output deviation $H_m - H$ at some temperatures has been provided by AMG:

H	-10°C	23°C	50°C	70°C	100°C	140°C
10%	-1%	0%	1%	2%	3%	4%
20%	-1.5%	0%	1.5%	3%	4.48%	6%
30%	-1.63%	0%	1.82%	3.55%	5.55%	7.43%
40%	-1.74%	0%	2.08%	4%	6.31%	8.35%
50%	-1.78%	0%	2.2%	4.2%	6.78%	8.78%
60%	-1.78%	0%	2.27%	4.35%	6.94%	9%
70%	-1.78%	0%	2.31%	4.43%	7%	9%
80%	-1.78%	0%	2.31%	4.51%	7.12%	9%
90%	-1.78%	0%	2.31%	4.51%	7.12%	9%

Since the desired circuit shall “compute” $H_m - H$ from H_m (and the temperature T), we plot the points $(H_m, H_m - H)$ for the above data:

6. Example Application



The first task was to find an algebraic function f_{spec} as specification of a “translinear synthesis” problem, so that these points satisfy (or are close to)

$$H_m - H = f_{\text{spec}}(H_m, T).$$

Some “educated guessing” and heavy usage of the MATHEMATICA-Functions `FindMinimum` and `ProjectiveRationalize` (from the standard package `NumberTheory‘Rationalize‘`) have found the function $f_{\text{spec}}(H_m, T) = \lambda(T)y(H_m)$, where

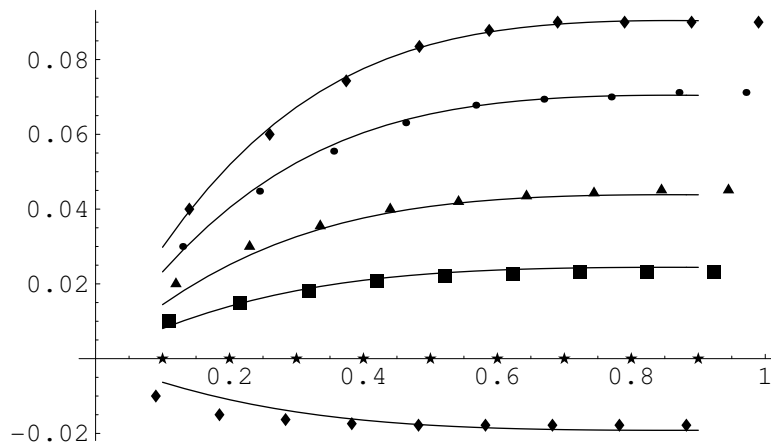
$$\lambda(T) = -0.0012167 (t - 10)(t - 1)(t + 3), \quad t = \frac{T}{23^\circ\text{C}},$$

$$y(H_m) = \frac{hH_m - H_m^2 + H_m\sqrt{5h^2 - 2hH_m + H_m^2}}{2h}, \quad (6.1)$$

and $h = 0.3294$. The function y is one of the two solutions of the implicit polynomial equation

$$hH_m^2 + hH_my - H_m^2y - hy^2 = 0. \quad (6.2)$$

The following plot shows the surprisingly high quality of the approximation:



6. Example Application

λ consists of simple multiplications and can be implemented easily by translinear as well as other analog circuitry, we don't go into further details of this part of the design problem.

Instead we concentrate on the synthesis of a subcircuit implementing y , where the tools developed in the framework of this thesis have been applied successfully.

The implicit description of the desired network behavior is given by (compare eqn. (6.2))

$$0 = g := u_1^2 u_2 + u_1 u_2 y - u_1^2 y - u_2 y^2.$$

The inputs are $u_1 = H_m$ and $u_2 = h$. (We use a stationary input current source for u_2 to supply the constant h to the network.) We wish to implement $g(u_1, u_2, y) = 0$ by a circuit with at most 6 transistors.

15 of the 24 translinear digraphs with 6 or less branches consist of only one loop of length 4 and some parallel branches, see Appendix A. For these 15 digraphs, the toric ideal has exactly one generator of degree 2 and no generator of higher degree. Consequently, for all formal networks $N = (G, C, v_0)$ where G is one of these digraphs, $\deg f_N = 2 < 3 = \deg g$, so the networks do not come into question for implementing g .

There are 9 digraphs remaining with 6 branches each. In the table of Appendix A, they carry the indices 6,7,8,9,10,19,20,21 and 24. A search for matches has been conducted through all formal networks based on one of these 9 digraphs. As bound for the entries of the input matrix, $l = 2$ was chosen.

For each of the 9 digraphs, the following table shows the number of pairs (N, B) of a formal translinear network N and an input matrix B which have been found. The second column shows the number of formal matches where $\varphi_B(f_N) = g$ (see Section 5.2), the third column shows how many of these passed the test for positive collector currents and correct explicit output function (see Section 5.4). The fourth column shows the approximate duration of the match search through all formal networks of the corresponding digraph (on a 1 GHz Pentium III processor). The duration of the final network check, in comparison, can be neglected. (For digraph 6, it took 0.2 seconds; for digraph 24, it took approximately 10 minutes.)

digraph index	pairs (N, B) with $\varphi_B(f_N) = g$	"relevant" pairs (N, B)	search duration
6	3	1	5 seconds
7	9	0	6.5 minutes
8	2	0	8 seconds
9	94	10	5.3 minutes
10	333	49	6.5 hours
19	40	9	8 seconds
20	870	167	2 days
21	387	0	5 seconds
24	1542	203	6 days

6. Example Application

Some of the networks have been simulated by E. J. Roebbers, employing transistor models that reflect the state-of-the-art IC technology of AMG. The simulations confirm that the networks indeed behave close to eqn. (6.1). At the time of writing, the found networks are awaiting more simulations and tests at AMG to decide which ones to adopt as candidates for hardware implementation.

7. Conclusion

The research reported in this thesis provides

- a coherent mathematical “translinear network theory”, consisting of a graph theoretic modelling framework for the topology of translinear circuits, and an analysis of the system of equations describing their behavior,
- the concept of a catalog of topologies for translinear networks as a resource for circuit design,
- algorithms to build such a catalog and to search it for a network complying with a particular behavior,
- and implementations of the algorithms, resulting in a software toolbox for translinear network synthesis.

As result, an exhaustive catalog of all static formal translinear networks with at most 8 transistors is available.

The details and implementations of the algorithms are worked out only for static networks, but can easily be adopted for dynamic networks as well.

While the implementation of the combinatorial algorithms is stand-alone software written “from scratch” in C++, the implementation of the algebraic algorithms, namely the symbolic treatment of the network equations and the match finding, heavily rely on the sophisticated Gröbner basis engine of SINGULAR and thus on more than a decade of experience contained in a special-purpose computer algebra system.

The application reported on in Chapter 6 proves the applicability of the developed synthesis approach.

It is worth to point out the new observation made by this thesis that the translinear loop equations of a network are precisely represented by the toric ideal of the network’s translinear digraph.

Altogether, the key role of translinear circuits as systematically designable nonlinear circuits is confirmed and strengthened.

Suggestions for Further Research

It is doubtful whether a catalog of dynamic translinear (DTL) network topologies could be an as effective tool as a catalog of only static networks topologies. The numbers of formal DTL networks will be much larger than for STL networks, so the sheer size of the catalog will grow to a severe obstruction, even if we restrict to networks with 1 or 2 capacitances.

However, the symbolic methods for match finding can be modified easily to include the determination of capacitance values as continuous parameters next to the discrete parameters the input matrix consists of.

Another direction of concept extension is the extension for MOS translinear circuits, see Section 3.5. If a suitable formulation for admissible drain connections can be found, the combinatorial and algebraic techniques of this thesis should be adaptable for a catalog of MOS translinear topologies.

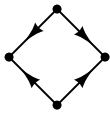
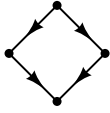
For academic interest one might try to prove the “principle ideal property” of static formal translinear networks (see Subsection 5.1.4), possibly with refined assumptions.

Of course, if the developed software tools appear to be of interest for a number of users, a proper user interface and an export link to circuit analysis software (for simulation or more detailed symbolic analysis) should be worked on. Efficiency could be improved by supplementing the match search routine with a suitable database management of the catalog.

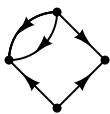
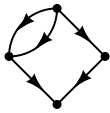
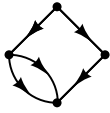
A. The Static Formal Translinear Networks with up to 6 Transistors

There are 2 translinear digraphs with 4 branches, 3 with 5 branches, and 19 with 6 branches. For each of these digraphs, the following tables show in the third column the number of different (with respect to generalized isomorphism, see Definition 4.7) valid collector assignments on G and in the fourth column the number of formal translinear networks (G, C, v_0) showing a nonlinear input-output relation.

Networks with 4 transistors:

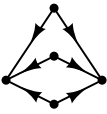
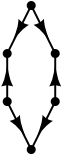
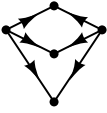
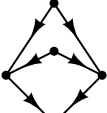
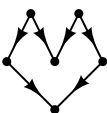
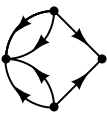
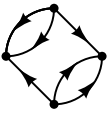
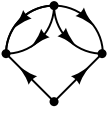
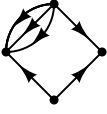
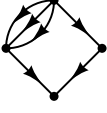
digraph index	translinear digraph	coll. assignments	formal networks
1		35	29
2		142	144

Networks with 5 transistors:

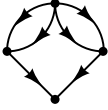
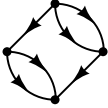
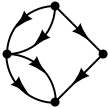
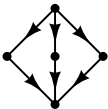
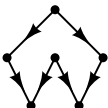
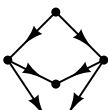
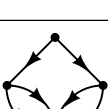
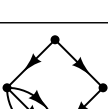
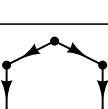
digraph index	translinear digraph	coll. assignments	formal networks
3		388	358
4		600	660
5		880	1083

A. The Static Formal Translinear Networks with up to 6 Transistors

Networks with 6 transistors (continued on next page):

digraph index	translinear digraph	coll. assignments	formal networks
6		300	291
7		537	398
8		413	628
9		1974	2593
10		4444	3757
11		661	577
12		661	727
13		661	727
14		959	910
15		1088	1235

A. The Static Formal Translinear Networks with up to 6 Transistors

digraph index	translinear digraph	coll. assignments	formal networks
16		754	867
17		2040	2608
18		2040	2608
19		2037	3471
20		7927	6499
21		2742	5073
22		1600	1829
23		2240	2848
24		9024	11602

B. Implementations

B.1. Overview of the Implementations

The algorithms presented in this thesis for building and searching a catalog of translinear network topologies have been implemented using C++, SINGULAR [GPS01], and MATHEMATICA [Wol99].

Implementations for *building* the catalog include:

- The combinatorial algorithms for the generation of static formal translinear networks as presented in Chapter 4.
The implementation is in C++, brief class descriptions and header file excerpts are found in Section B.2. The implementation comprises two executables:
 - `graphgen`, a little tool for generating translinear digraphs,
 - `tlgen`, the main program for the generation of formal networks.
- The elimination of collector currents from the system of network equations, see Subsection 5.1.4.
The output of `tlgen` is one file of SINGULAR code for each translinear digraph. A sample is included in Section B.3. If executed, the SINGULAR code performs the elimination of collector currents, and, in those cases where the elimination yields nonlinear equations (see Remark 5.5), writes the results in a file of “.mp” format¹.

Thus the catalog consists of one .mp file for each translinear digraph. For each collector assignment on the respective digraph, the .mp file contains an ideal representing the system of network equations where the collector currents have been eliminated. Since a ground node is not yet specified at this point, node equations for all nodes of G are used for this system.

On a 2 GHz Athlon XP 2800 processor, `tlgen` takes about 10 hours to generate the formal networks with up to 8 transistors. The generation of formal networks with up to 7 transistors is finished after 15 minutes, the generation of networks with up to 6 transistors after 1 minute.

¹See the SINGULAR documentation for this so-called *MultiProtocol* format.

B. Implementations

Implementations for *using* the catalog include:

- A SINGULAR procedure `matchfind`, for testing a network polynomial f_N for a match onto a “target behavior polynomial” g , see Section 5.3. The code of this procedure is listed in Section B.4.
- A SINGULAR procedure `netsearch` which searches the formal networks based on a particular translinear digraph for a network complying with a given behavior. As parameters, it takes a “target polynomial” g , a digraph identification index, and a bound for the entries of the input matrix. `netsearch` reads the `.mp` file corresponding to the identified digraph. The code of `netsearch` is not listed in this appendix, because it consists mainly of bookkeeping. The functional tasks it fulfills are to iterate through the collector assignments C , for each C to successively pick possible ground nodes v_0 , for each v_0 to eliminate the input current variable i_{v_0} from the pre-processed network equations and thus determine the network polynomial f_N , and then to call `matchfind`.
`netsearch` writes an output file in MATHEMATICA language containing the topology of the found networks, various information about their systems of equations, and the determined input matrices.
- A MATHEMATICA function `postest` which performs the final network checks for positive collector currents and for the correct output function, as presented in Section 5.4. The MATHEMATICA code is listed in Section B.5.

B.2. Combinatorial Generation of Formal Networks

B.2.1. The class “matrix”

The objects of this class represent integer matrices and are used for adjacency matrices of digraphs.

```
class matrix
{
protected:
  ubyte rows;
  ubyte cols;
  ubyte * data;
  int cmp_permute(const permutation&, const permutation&) const;
public:
  matrix(const ubyte& value);
  matrix(const ubyte&, const ubyte&);
  matrix(const matrix&);
  matrix& operator =(const matrix&);
  virtual ~matrix();
  void set_entry(ubyte row, ubyte col, ubyte value)
    { data[ row * cols + col ] = value; }
  ubyte rs() const { return rows; }
  ubyte cs() const { return cols; }

  // permute rows and columns
  matrix permute(const permutation& rowperm, const permutation& colperm) const;

  bool is_zero() const;
  void add_zero_row();
  void add_zero_col();

  // test whether the matrix is canonical w.r.t. the given group of column
  // permutations and all row permutations
  bool is_canonical(const mpsims& colgroup) const;

  // test whether a pair of a row permutation and a column permutation is
  // an automorphism, i.e. does not change the matrix
  bool test_automorphism(const permutation& rp, const permutation& cp) const;

  friend istream& operator >>(istream &, matrix &);
  friend ostream& operator <<(ostream &, const matrix &);
  friend class ranked;
  friend class edge_coloring;
  friend ostream& operator <<(ostream &, const ranked &);
};
```

B.2.2. The classes “permutation” and “multiperm”

The objects of the class “permutation” represent elements of S_n . The objects of the class “multiperm” represent elements of $S_{n_1} \times \cdots \times S_{n_R}$.

```
class permutation : public vector<ubyte>
{
private:
    permutation() : vector<ubyte>() {}
public:
    permutation(const ubyte& size);

    // compute the inverse permutation
    permutation inv() const;

    // product/concatenation of permutations
    permutation operator * (const permutation& perm) const;
};

class multiperm : public vector<permutation>
{
private:
    multiperm() {}
public:
    static const multiperm NIL;
    multiperm(const vector<ubyte>& order);

    // inverse and product of multiperms...
    multiperm inv() const;
    void mult_left(const multiperm& mp)
    {
        transform(mp.begin(),mp.end(),begin(),begin(),multiplies<permutation>());
    }
    void mult_right(const multiperm& mp)
    {
        transform( begin(), end() ,mp.begin(),begin(),multiplies<permutation>());
    }

    // dimension vector
    vector<ubyte> order() const
    {
        ...
    }
};
```


B.2.3. The class “mpsims”

An object of this class represents a subgroup of $S_{n_1} \times \cdots \times S_{n_R}$. The class name comes from the data structure called *sims chains* [KS99, Jer86], which is used to store the subgroup.

```

class mpsims // sims chain of multiperms
            // represents subgroup G_1 x ... x G_n of S_1 x ... x S_n
{
private:
    typedef vector<multiperm> transversal;
    vector<ubyte> order;
    vector<transversal> sims;
    mpsims();
public:
    // build identity sims chain
    mpsims(const vector<ubyte>& _order);

    // enclose a multiperm with the precondition that it will only affect the
    // two top ranks
    void enclose_2top(multiperm);

    // helper class for iterating through the elements of G_n
    class top_iterator;

    // restrict to the n-1 non-top ranks
    void decap();

    // cross product with the trivial permutation group on nr points
    void trivially_extend(ubyte nr);

    // cross product with the maximal G_(n+1) such that
    // G_n x G_(n+1) consists of automorphisms
    void auto_extend(const matrix&);

    friend ostream& operator <<(ostream& out, const mpsims& chain);
};

class mpsims::top_iterator
{
    ...
};

```

B.2.4. The class “ranked”

An object of this class represents a layered (or “ranked”) digraph.

```
class ranked // representations of layered digraphs
{
private:
    vector<matrix> adj; // adjacency matrices
    // The k-th matrix is the matrix from potential k to potential k+1.
    // (note the case k=0!)
    bool test_leaf(ubyte node) const;
public:
    ranked(const composition&);
    ranked(const matrix&);

    // number of nodes
    int vertices() const;

    // loop equations for a system of fundamental loops
    string tl_equations() const;

    ...

    // compute augmentations, write them in the given queue
    // max_parallel is an optional bound for the matrix entries
    mpsims augment(queue<ranked>&, ubyte max_parallel = 0) const;

    // compute automorphism group
    mpsims autogroup() const;

    // how many nodes in each layer
    vector<ubyte> dim() const;

    ubyte entry(ubyte rank, ubyte row, ubyte col) const;
    friend istream& operator >> (istream& in, ranked&);
    friend ostream& operator << (ostream& out, const ranked&);
};
```

B.2.5. The class “network”

An object of this class represents a collector assignment on a translinear digraph. (In the sense of Definition 3.17, it is not a complete network description, since it lacks information about the ground node. However, given a network object of this class, we know G and C , and it is trivial to iterate through all ground nodes $v_0 \in V(G) \setminus \text{img}(C)$.)

```
// some auxiliary classes...

class multinode : public vector<ubyte>
{
private:
public:
    multinode(ubyte cardinality) : vector<ubyte>(cardinality) {}
    ubyte& node_at(ubyte index) { return operator[](index); }
    const ubyte& node_at(ubyte index) const { return operator[](index); }
    void zero() { fill(begin(),end(),0); }
    void zero_C(ubyte min_rank);
    bool inc(ubyte min_rank, const vector<ubyte>& dim);
    bool inc_out();
    bool inc_C(ubyte min_rank, const vector<ubyte>& dim);
    ubyte outsign() const;
    int compare(const multinode&) const;
    multinode apply_mp(ubyte max_rank,const multiperm&) const;
};

class mnmat : public vector<multinode> // mnmat = multinode matrix
{
public:
    int permcmp(ubyte max_rank,const multiperm&) const;
    int permcmp(ubyte max_rank, multiperm, const permutation&) const;
    bool inc(ubyte min_rank, const vector<ubyte>& dim);
    bool inc_C(ubyte min_rank, const vector<ubyte>& dim);
    bool inc_out();
};

inline bool operator < (multinode m1, multinode m2)
{
    if(m1.size()!=m2.size())
        throw err("< :multinodes must have same cardinality");
    else
        return lexicographical_compare(m1.begin(),m1.end(),m2.begin(),m2.end());
}
```

B. Implementations

```
//  
// NETWORK CLASS  
//  
class network {  
private:  
    vector<ubyte> dim;  
    vector<mmat> c; // c = collector connections  
    mutable set<ubyte> gnodes;  
public:  
    network(const ranked& graph); // "zero-network": all collectors are outputs  
    // note that the graph is flipped (top rank becomes 0-rank)  
  
    // compute "the next" collector assignment, 3 variations  
    // return true iff "next" exists  
    bool inc();  
    bool inc_out();  
    bool inc_C();  
  
    // test for canonicity  
    bool is_canonical(mpsims g) const;  
  
    // test for generalized canonicity  
    bool is_gcanonical(mpsims g) const;  
  
    // complement of the image of the collector assignment  
    // groundnodes() only works if node_equations() has been called before  
    set<ubyte> groundnodes() { return(gnodes); }  
  
    string transistor_list() const;  
    string node_equations() const;  
    string internal_vars() const;  
    string output_vars() const;  
    friend ostream& operator <<(ostream&, const network&);  
};
```

B.3. Sample tlgcn Output

The following Singular code is the beginning and the end of the tlgcn output file for the translinear digraph with index 1 (see Appendix A).

```

int count;
proc testlin (ideal j)
{
int is_linear = 1;
for(int k=1; k<=size(j); k++) { if( deg(j[k]) != 1 ) { is_linear=0;}}
if (is_linear==0) { count++; }
return(is_linear);
}

ring r = 0, (y,i(10),i(11),i(00),i(01),x(000),x(001),x(010),x(011) ), dp;
ideal id_nodes, id_net, id_e, lindep_nodes;
int islin;
list outvarlist;
string outfile="1.mp";
link l="MPfile:w " + outfile;
write(l,r);
"sng/1";
poly xprod=x(000)* x(010)* x(001)* x(011);
ideal id_graph = x(011)* x(000) - x(010)* x(001) ,
0;
LIB "elim.lib";
id_graph = sat(id_graph,xprod)[1];
write(l,id_graph);
list groundnodes;
id_nodes = -y +x(011),
- i(10),
- i(11) +x(000),
- i(00) -x(000) -x(001),
- i(01) -x(010) -x(011)
;
lindep_nodes = eliminate(id_nodes,xprod);
id_net = id_graph,id_nodes;
id_e = eliminate(id_net,xprod);
id_e;
islin = testlin(id_e);
if( islin == 0)
{
write(l,"TransistorList[
{ Emitter[0,0], Base[1,0], Collector[1,1]},
{ Emitter[0,1], Base[1,0], Collector[void]},
{ Emitter[0,0], Base[1,1], Collector[void]},

```

B. Implementations

```
{ Emitter[0,1], Base[1,1], Collector[outp]}}");
write(l,id_nodes);
write(l,id_e);
groundnodes =list();
if(eliminate(lindep_nodes,i(10)) == 0 )
{ groundnodes= insert(groundnodes,i(10)); }
write(l,groundnodes);
}
id_nodes = -y +x(001),
- i(10),
- i(11) +x(000),
- i(00) -x(000) -x(001),
- i(01) -x(010) -x(011)
;
lindep_nodes = eliminate(id_nodes,xprod);
id_net = id_graph,id_nodes;
id_e = eliminate(id_net,xprod);
id_e;
islin = testlin(id_e);
if( islin == 0)
{
write(l,"TransistorList[
{ Emitter[0,0], Base[1,0], Collector[1,1]},
{ Emitter[0,1], Base[1,0], Collector[void]},
{ Emitter[0,0], Base[1,1], Collector[outp]},
{ Emitter[0,1], Base[1,1], Collector[void]}}");
write(l,id_nodes);
write(l,id_e);
groundnodes =list();
if(eliminate(lindep_nodes,i(10)) == 0 )
{ groundnodes= insert(groundnodes,i(10)); }
write(l,groundnodes);
}
id_nodes = -y +x(001) -x(011),
- i(10),
- i(11) +x(000),
- i(00) -x(000) -x(001),
- i(01) -x(010) -x(011)
;
lindep_nodes = eliminate(id_nodes,xprod);
id_net = id_graph,id_nodes;
id_e = eliminate(id_net,xprod);
id_e;
islin = testlin(id_e);
if( islin == 0)
{
```

B. Implementations

```
    write(l,"TransistorList[
{ Emitter[0,0], Base[1,0], Collector[1,1]},
{ Emitter[0,1], Base[1,0], Collector[void]},
{ Emitter[0,0], Base[1,1], Collector[outp]},
{ Emitter[0,1], Base[1,1], Collector[outm]}}");
    write(l,id_nodes);
    write(l,id_e);
    groundnodes =list();
    if(eliminate(lindep_nodes,i(10)) == 0 )
    { groundnodes= insert(groundnodes,i(10)); }
    write(l,groundnodes);
}

...

id_nodes = -y +x(011),
- i(10),
- i(11) +x(000) +x(010) +x(001),
- i(00) -x(000) -x(001),
- i(01) -x(010) -x(011)
;
lindep_nodes = eliminate(id_nodes,xprod);
id_net = id_graph,id_nodes;
id_e = eliminate(id_net,xprod);
id_e;
islin = testlin(id_e);
if( islin == 0)
{
    write(l,"TransistorList[
{ Emitter[0,0], Base[1,0], Collector[1,1]},
{ Emitter[0,1], Base[1,0], Collector[1,1]},
{ Emitter[0,0], Base[1,1], Collector[1,1]},
{ Emitter[0,1], Base[1,1], Collector[outp]}}");
    write(l,id_nodes);
    write(l,id_e);
    groundnodes =list();
    if(eliminate(lindep_nodes,i(10)) == 0 )
    { groundnodes= insert(groundnodes,i(10)); }
    write(l,groundnodes);
}
write(l,"EOF!");
close(l);
"72 (total number of networks)";
"35 (number of valid networks)";
"number of effectively non-linear networks:";
count;
```

B.4. Match Finding

The following SINGULAR procedure is used to check a network polynomial f_N for a match onto a polynomial g (see Section 5.3).

```
LIB "primdec.lib";

proc matchfind (poly netpoly, poly g, int n_u, int n_c,
               int coefbound, int modulus, ideal addconstraints)
{
  def rc=basing;
  int j,k;
  poly intconstraint = c(1);
  for( j = 1; j<=coefbound; j++ )
  {
    intconstraint = intconstraint*(c(1)-j)*(c(1)+j);
  }
  ideal intconstraints = addconstraints;
  for ( j = 1; j<=n_c; j++ )
  {
    intconstraints = intconstraints,subst(intconstraint,c(1),c(j));
  }
  poly uymon=y;
  for ( j=1; j <= n_u; j++ )
  {
    intconstraints = intconstraints,subst(intconstraint,c(1),b(j));
    uymon = uymon * u(j);
  }
  matrix netcoef = coef(netpoly,uymon);
  matrix gcoef = coef(g,uymon);

  list solutions = list();
  poly zero = dummy1*g - netpoly;
  matrix zerocoeff=coef(zero,uymon);
  ideal sdum=std(dummy1);
  for( j=1; j<=ncols(zerocoeff); j++)
  {
    if( reduce(zerocoeff[2,j],sdum) == 0 )
    {
      return(list());
    }
  }
  ideal coefcompare = zerocoeff[2,1..ncols(zerocoeff)];
  coefcompare = intconstraints,coefcompare,dummy1*dummy2-1;

  ideal p;
}
```


B. Implementations

```
ring modring=modulus,(c(1..n_c),b(1..n_u),dummy1,dummy2),dp;
ideal coefcompare = imap(rc,coefcompare);
coefcompare=std(coefcompare);
list l=minAssGTZ(coefcompare);
ideal idprime;
for( j=1; j <= size(l); j++)
{
  idprime = interred(l[j]);
  setring rc;
  p = imap(modring,idprime),intconstraints,coefcompare;
  p = std(p);
  if( p[1] != 1 )
  {
    solutions = insert(solutions,eliminate(p,dummy1*dummy2));
  }
  setring modring;
}
setring rc;
return(solutions);
}
```

B.5. Final Network Checks

The following MATHEMATICA code is used for the final network check (see Section 5.4), incorporating the positivity test for the collector currents and the output function check.

```

allposQ[terms_List]:=Select[terms,Simplify[#>0,assump]!=True& ,1];
postest:=Module[{l},
  l=Read[inp,Expression];
  found=0;
  While[Not[l===EndOfFile],
    l=l/.{u[1]->U,u[2]->H};
    eqs=l[[4]];
    positives=Select[Solve[eqs,colcurs],allposQ[colcurs/.#]=={ }&];
    If[Length[positives]>1,
      Print["???"];
      Exit[]
    ];
    If[ positives != { } && Simplify[func ==
      y/.Solve[(l[[5]]/.positives[[1]]/.l[[3,1]]) == 0,y] [[1]]]
    ,
      found++;
      Print["Network number: ",l[[1]]];
      Print["ground node: ",l[[2]]];
      Print["interface: ",l[[3]]];
      Print["netlist: ",l[[6]]];
      Print["collector currents: ",positives[[1]]];
    ];
    l=Read[inp,Expression];
  ];
Close[inp];
Print["found ",found," networks"];
];

```

Bibliography

- [AB96] Andreas G. Andreou and Kwabena A. Boahen. Translinear circuits in subthreshold MOS. *Analog Integrated Circuits and Signal Processing*, 9:131–166, 1996.
- [AGB65] K. N. Stevens A. G. Bose. *Introductory Network Theory*. Harper Row, New York, 1965.
- [ai01] <http://www.analog-insydes.de>, 2001.
- [AK92] E. F. Assmus and J. D. Key. *Designs and their codes*, volume 103 of *Cambridge tracts in mathematics*. Cambridge Univ. Pr., Cambridge, 1992.
- [AL94] William W. Adams and Philippe Loustau. *An Introduction to Gröbner Bases*, volume 3 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1994.
- [And90] Ian Anderson. *Combinatorial designs – construction methods*. Ellis Horwood series in mathematics and its applications. Horwood, Chichester, 1990.
- [BBS69] Norman Balabanian, Theodore A. Bickart, and Sundaram Seshu. *Electrical Network Theory*. Wiley, New York, 1969.
- [BFK⁺98] Anton Betten, Harald Friepertinger, Adalbert Kerber, Alfred Wassermann, and Karl-Heinz Zimmermann. *Codierungstheorie. Konstruktion und Anwendung linearer Codes*. Springer, Berlin, Heidelberg, 1998.
- [BJL99] Thomas Beth, Dieter Jungnickel, and Hanfried Lenz. *Design theory*, volume 69 and 78 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, Cambridge, 2nd edition, 1999.
- [BLL98] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial Species and Tree-like Structures*, volume 67 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, Cambridge, 1998.
- [Bri85] Egbert Brieskorn. *Lineare Algebra und Analytische Geometrie II*. Vieweg, Braunschweig, 1985.

Bibliography

- [BS68] R. G. Busacker and T. L. Saaty. *Endliche Graphen und Netzwerke*. Oldenbourg, München, Wien, 1968.
- [BW93] Thomas Becker and Volker Weispfennig. *Gröbner Bases – A computational approach to commutative algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, Heidelberg, 1993.
- [CDK87] L. O. Chua, C. A. Desoer, and E. S. Kuh. *Linear and Nonlinear Circuits*. McGraw-Hill, New York, 1987.
- [Che71] Wai-Kai Chen. *Applied Graph Theory*, volume 13 of *North-Holland Series in Applied Mathematics and Mechanics*. North-Holland Publishing Company, Amsterdam, 1971.
- [CLO97] David Cox, John Little, and Donal O’Shea. *Ideals, Varieties, and Algorithms*. Springer-Verlag, New York, Berlin, Heidelberg, 2. edition, 1997.
- [Col96] Charles J. Colbourn, editor. *The CRC handbook of combinatorial designs*. The CRC Press series on discrete mathematics and its applications. CRC Press, Boca Raton, Fla., 1996.
- [dB64] N. G. de Bruijn. Polya’s theory of counting. In Edwin F. Beckenbach, editor, *Applied Combinatorial Mathematics*, chapter 5, pages 144–184. Wiley, New York, 1964.
- [DBS04] Ying Dong, Sumit Bagga, and Wouter A. Serdijn. An inherently linear CMOS multiplier. In *Proc. ProRISC 2004*, pages 483–486, Veldhoven, November 2004. http://www.stw.nl/prorisc/proc-2004/analog_circuits/dong.pdf.
- [Dem97] Peter Dembowski. *Finite geometries*. Classics in mathematics. Springer, Berlin, repr. of the 1968 ed. edition, 1997.
- [DK69] Charles A. Desoer and Ernest S. Kuh. *Basic Circuit Theory*. McGraw-Hill Kogakusha, Tokyo, 1969.
- [dLST95] Jesús A. de Loera, Bernd Sturmfels, and Rekha R. Thomas. Gröbner bases and triangulations of the second hypersimplex. *Combinatorica*, 15(3):409–424, 1995.
- [DRVRV99] Manuel Delgado-Restituto, Fernando Vidal, and Angel Rodríguez-Vázquez. Nonlinear circuit synthesis using integrated circuits. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, volume 14 [Mu-Nu], pages 472–502. Wiley, New York, 1999.
- [DST92] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra – Systems and algorithms for algebraic computations*. Academic Press, London, 2. edition, 1992.

Bibliography

- [Ehr65] Charles Ehresmann. *Catégories et structures*, volume 10 of *Collection travaux et recherches mathématiques*. Dunod, Paris, 1965.
- [Eis96] David Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, Heidelberg, 1996.
- [Fou92] L. R. Foulds. *Graph Theory Applications*. Springer-Verlag, New York, Berlin, Heidelberg, 1992.
- [FS83] Gerd Fischer and Reinhard Sacher. *Einführung in die Algebra*. Teubner, Stuttgart, 3rd edition, 1983.
- [Ful97] William Fulton. *Introduction to toric varieties*, volume 131 of *Annals of mathematics studies*. Princeton Univ. Pr., Princeton, NJ., 1997. 2. corr. print.
- [GCL92] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Boston, 1992.
- [Gil68] Barrie Gilbert. A new wide-band amplifier technique. *IEEE Journal of Solid-State Circuits*, SC-3(4):353–365, December 1968.
- [Gil75] Barrie Gilbert. Translinear circuits: A proposed classification. *Electronics Letters*, 11(1):14–16, January 1975.
- [Gil83] Barrie Gilbert. A four-quadrant analog divider/multiplier with 0.01% distortion. In *IEEE International Solid-State Circuits Conference*, 1983.
- [Gil90] Barrie Gilbert. Current-mode circuits from a translinear viewpoint: A tutorial. In C. Toumazou, F. J. Lidgey, and D. G. Haigh, editors, *Analogue IC design: The Current-mode Approach*, volume 2 of *IEE Circuits and Systems Series*, kapitel 2. Peter Peregrinus, London, 1990.
- [Gil96] Barrie Gilbert. Translinear circuits: An historical overview. *Analog Integrated Circuits and Signal Processing*, 9(2):95–118, March 1996.
- [Gil99] Barrie Gilbert. Translinear circuits. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, volume 22 [Th-Un], pages 442–461. Wiley, New York, 1999.
- [GJ83] Ian P. Goulden and David M. Jackson. *Combinatorial Enumeration*. Wiley-Interscience series in discrete mathematics. Wiley, New York, 1983.
- [GK79] R. Genin and R. Konn. Sinusoidal frequency doubler. *Electronic Letters*, 15(2):47–48, January 1979.

Bibliography

- [GLM97] Thomas Grüner, Reinhard Laue, and Markus Meringer. Algorithms for group actions applied to graph generation. In Larry Finkelstein and William M. Kantor, editors, *Groups and Computation II*, volume 28 of *DIMACS Series in Discrete Mathematics and Computer Science*, pages 113–123. AMS, 1997.
- [GM92] Paul R. Gray and Robert G. Meyer. *Analysis and Design of Analog Integrated Circuits*. Wiley, New York, 3. edition, 1992. insbes. Abschn. 2.5, Active Devices in Bip. An. ICs.
- [GP02] Gert-Martin Greuel and Gerhard Pfister. *A SINGULAR introduction to commutative algebra*. Springer, Berlin, 2002.
- [GPS01] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 2.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2001. <http://www.singular.uni-kl.de>.
- [Gru93] Roland Grund. *Konstruktion schlichter Graphen mit gegebener Gradpartition*, volume 44 of *Bayreuther Math. Schriften*, pages 73–104. Universität Bayreuth, 1993.
- [GS91] G. Gielen and W. Sansen. *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Kluwer Academic Publishers, Boston, 1991.
- [Gui49] E. A. Guillemin. *The Mathematics of Circuit Analysis*. Wiley, New York, 1949.
- [Har67] Frank Harary. *Graph theory and theoretical physics*. Academic Press, London, 1967.
- [Hea72] B. R. Heap. The production of graphs by computer. In Ronald C. Read, editor, *Graph Theory and Computing*, pages 47–62. Academic Press, New York, 1972.
- [HH73] Douglas J. Hamilton and William G. Howard. *Basic Integrated Circuit Engineering*. McGraw-Hill Kogakusha, Tokyo, 1973.
- [HKL⁺91] R. Hager, A. Kerber, R. Laue, D. Moser, and W. Weber. *Construction of orbit representatives*, volume 35 of *Bayreuther Math. Schriften*, pages 157–169. Universität Bayreuth, 1991.
- [HS02] Sandro A. P. Haddad and Wouter A. Serdijn. High-frequency dynamic translinear and log-domain circuits in cmos technology. In *proc. IEEE International Symposium on Circuits and Systems*, pages 26–29, 2002.
- [Hun74] Thomas W. Hungerford. *Algebra*. Holt, Rinehart and Winston, New York, 1974.

Bibliography

- [II00a] Takayuki Ishizeki and Hiroshi Imai. Complexity of gröbner bases for toric ideals of acyclic tournament graphs. *RIMS Kokyuroku, Kyoto University*, 1148:134–139, 2000.
- [II00b] Takayuki Ishizeki and Hiroshi Imai. Gröbner bases for toric ideals of acyclic directed graphs and their applications to minimum cost flow problem. In *Optimization—Modeling and Algorithms 14—*, pages 152–165. Institute of Statistical Mathematics Cooperative Research Report 135, 2000.
- [Ils02] David Ilsen. Algebraische Aspekte der Synthese translinearer Netzwerke. Diplomarbeit, Universität Kaiserslautern, April 2002.
- [Ils04] David Ilsen. Cataloging translinear networks. In *Proc. 8th International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD 2004)*, pages 32–35, Wroclaw, September 2004.
- [IR04] David Ilsen and Ernst Josef Roebbers. Translinear networks from a combinatorial viewpoint. In *Proc 15th ProRISC Workshop*, volume 11, pages 524–528, November 2004.
- [Iri69] Masao Iri. *Network Flow, Transportation and Scheduling*, volume 57 of *Mathematics in Science and Engineering*. Academic Press, New York, 1969.
- [Ish00a] Takayuki Ishizeki. Analysis of gröbner bases for toric ideals of acyclic tournament graphs. Master’s thesis, Department of Information Science, University of Tokyo, 2000.
- [Ish00b] Takayuki Ishizeki. Gröbner bases of acyclic directed graphs and reductions in conti-transverso algorithm. *RIMS Kokyuroku, Kyoto University*, 1175:63–71, 2000.
- [Jer86] Mark Jerrum. A compact representation for permutation groups. *J. Algorithms*, 7:60–78, 1986.
- [Joy81] André Joyal. Une théorie combinatoire des séries formelles. *Advances in Mathematics*, 42:1–82, 1981.
- [Jun99] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Springer, Berlin, 1999.
- [KBN01] Eric A. M. Klumperink, Federico Bruccoleri, and Bram Nauta. Finding all elementary circuits exploiting transconductance. *IEEE Transactions on Circuits and Systems II*, 48(11):1039–1053, November 2001.
- [Ker99] Adalbert Kerber. *Applied Finite Group Actions*. Springer, Berlin, second edition, 1999.

Bibliography

- [Klu97] Eric A. M. Klumperink. *Transconductance Based CMOS Circuits*. PhD thesis, University of Twente, 1997.
- [KS99] Donald L. Kreher and Douglas R. Stinson. *Combinatorial Algorithms – Generation, Enumeration, and Search*. The CRC Press series on discrete mathematics and its applications. CRC Press, Boca Raton, 1999.
- [KT83] A. Kerber and K.-J. Thürlings. *Symmetrieklassen von Funktionen und ihre Abzählungstheorie. I: Die Grundprobleme*, volume 12 of *Bayreuther Math. Schriften*. Universität Bayreuth, 1983.
- [Lan94] Serge Lang. *Algebra*. Addison-Wesley, Reading, Mass., 3. edition, 1994.
- [Lau89] Reinhard Laue. *Eine konstruktive Version des Lemmas von Burnside*, volume 28 of *Bayreuther Math. Schriften*, pages 111–125. Universität Bayreuth, 1989.
- [Lau93] Reinhard Laue. *Construction of Combinatorial Objects – A Tutorial*, volume 43 of *Bayreuther Math. Schriften*, pages 53–96. Universität Bayreuth, 1993.
- [Lau99] Reinhard Laue. Constructing objects up to isomorphism, simple 9-designs with small parameters. In Anton Betten, Axel Kohnert, Reinhard Laue, and Alfred Wassermann, editors, *Algebraic Combinatorics and Applications*, pages 232–260, Berlin; Heidelberg, September 1999. Springer.
- [Liu77] C. L. Liu. *Elements of Discrete Mathematics*. McGraw-Hill, New York, 1977.
- [LOC75] P.-M. Lin L. O. Chua. *Computer-Aided Analysis of Electronic Circuits: Algorithms & Computational Techniques*. Prentice Hall, Englewood Cliffs, NJ, 1975.
- [Mat87] W. Mathis. *Theorie nichtlinearer Netzwerke*. Springer-Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1987.
- [MHvL73] Jr. M. Hall and J. H. von Lint, editors. *Combinatorics*, volume 2. Mathematical Centre, Amsterdam, 1973.
- [Min97] Bradley A. Minch. *Analysis, Synthesis, and Implementation of Networks of Multiple-Input Translinear Elements*. PhD thesis, California Institute of Technology, Pasadena, Kalifornien, 1997.
- [MSvdWvR98] Jan Mulder, Wouter A. Serdijn, Albert C. van der Woerd, and Arthur H. M. van Roermund. An instantaneous and syllabic companding translinear filter. *IEEE Transactions on Circuits and Systems I*, 45(2):150–153, 1998.

Bibliography

- [MSvdWvR99] Jan Mulder, Wouter A. Serdijn, Albert C. van der Woerd, and Arthur H. M. van Roermund. *Dynamic Translinear and Log-Domain Circuits – Analysis and Synthesis*. Kluwer Academic Publishers, Boston, 1999.
- [MvdWSvR95] Jan Mulder, Albert C. van der Woerd, Wouter A. Serdijn, and Arthur H. M. van Roermund. Application of the back gate in MOS weak inversion translinear circuits. *IEEE Transactions on Circuits and Systems I*, 42(11):958–962, Nov 1995.
- [MvdWSvR97] Jan Mulder, Albert C. van der Woerd, Wouter A. Serdijn, and Arthur H. M. van Roermund. General current-mode analysis method for translinear filters. *IEEE Transactions on Circuits and Systems I*, 44(3):193–197, 1997.
- [OH99] Hidefumi Ohsugi and Takayuki Hibi. Toric ideals generated by quadratic binomials. *Journal of Algebra*, 218:509–527, 1999.
- [OM99] Kohshi Okumura and Hideki Morino. Application of groebner bases to analysis of translinear circuits. In *7th International Workshop on Nonlinear Dynamics of Electronic Systems*, 1999.
- [Pat63] William L. Paterson. Multiplication and logarithmic conversion by operational amplifier-transistor circuits. *Review of Scientific Instruments*, 34(12):1311–1316, December 1963.
- [Pól37] George Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und Chemische Verbindungen. *Acta math.*, 68:145–254, 1937.
- [PR87] George Pólya and Ronald C. Read. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*. Springer, New York, 1987.
- [Rea78] Ronald C. Read. *Every one a winner*, volume 2 of *Ann. Discrete Math.*, pages 107–120. North-Holland, Amsterdam, 1978.
- [Red27] J. H. Redfield. The theory of group-reduced distributions. *Amer. J. Math.*, 49:433–455, 1927.
- [RVDRHV95] A. Rodríguez-Vázquez, M. Delgado-Restituto, J. L. Huertas, and Fernando Vidal. Synthesis and design of nonlinear circuits. In Wai-Kai Chen, editor, *The Circuits and Filters Handbook*, chapter 32, pages 935–972. CRC Press, IEEE Press, 1995.
- [Sch02] Jürgen Schmitz. Systematic generation of VCCS circuit topologies with cospectral and isomorphic multigraphs. In *Proc. 7th International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD 2004)*, pages 25–30, Sinaia, October 2002.

Bibliography

- [Sch04] Jürgen Schmitz. On determining "useful" VCCS circuit topologies. In *Proc. 8th International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD 2004)*, pages 52–55, Wrocław, September 2004.
- [See88] Evert Seevinck. *Analysis and Synthesis of Translinear Circuits*, volume 31 of *Studies in Electrical and Electronic Engineering*. Elsevier, Amsterdam, 1988.
- [See90] Evert Seevinck. Companding current-mode integrator: A new circuit principle for continuous-time monolithic filters. *Electronic Letters*, 26(24):2046–2047, November 1990.
- [Ser05] Wouter A. Serdijn. Personal Communication, April 2005.
- [SGLBA99] Teresa Serrano-Gotarredona, Bernabé Linares-Barranco, and Andreas G. Andreou. A general translinear principle for subthreshold MOS transistors. *IEEE Transactions on Circuits and Systems I*, 46(5):607–616, May 1999.
- [SH98] R. Sommer and E. Hennig. Rechnergestützte Berechnung elektrischer Schaltungen. Lecture notes, Universität Kaiserslautern, 1995-1998.
- [Ski91] Steven Skiena. *Implementing Discrete Mathematics – Combinatorics and Graph Theory with Mathematica*. Addison-Wesley, Redwood City, Calif., 1991.
- [SKMvR99] Wouter A. Serdijn, Michiel H.L. Kouwenhoven, Jan Mulder, and Arthur H. M. van Roermund. Design of high dynamic range fully integrable translinear filters. *Analog Integrated Circuits and Signal Processing*, 19:223–239, 1999.
- [Sle68] Paul Slepian. *Mathematical Foundations of Network Analysis*, volume 16 of *Springer Tracts in Natural Philosophy*. Springer-Verlag, Berlin, Heidelberg, 1968.
- [SMvdWvR98] Wouter A. Serdijn, Jan Mulder, Albert C. van der Woerd, and Arthur H. M. van Roermund. A wide-tunable translinear second-order oscillator. *IEEE J. of Solid-State Circuits*, 33(2):195–201, 1998.
- [SR61] S. Seshu and M. B. Reed. *Linear graphs and electrical networks*. Reading, Mass., 1961.
- [ST81] M. N. S. Swamy and K. Thulasiraman. *Graphs, Networks, and Algorithms*. Wiley, New York, 1981.
- [Sta96] Richard P. Stanley. *Combinatorics and Commutative Algebra*, volume 41 of *Progress in mathematics*. Birkhäuser, Boston, 2nd edition, 1996.

Bibliography

- [Stu97] Bernd Sturmfels. *Gröbner Bases and Convex Polytopes*, volume 8 of *University lecture series*. American Mathematical Soc., Providence, 1997.
- [SVV94] Aron Sims, Wolmer V. Vasconcelos, and Rafael H. Villareal. On the ideal theory of graphs. *Journal of Algebra*, 167:389–416, 1994.
- [SW86] Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer, New York, 1986.
- [The99] Christine Theis. Der Buchberger-Algorithmus für torische Ideale und seine Anwendung in der ganzzahligen Optimierung. Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1999.
- [Tuc84] Alan Tucker. *Applied Combinatorics*. Wiley, New York, 2nd edition, 1984.
- [Val60] M. E. Valkenburg. *Introduction to modern network synthesis*. Wiley, New York, 1960.
- [VBSRS00] Rafael Vargas-Bernal, Arturo Sarmiento-Reyes, and Wouter A. Serdijn. Identifying translinear loops in the circuit topology. In *proc. IEEE International Symposium on Circuits and Systems II*, pages 585–588, 2000.
- [vV64] M. E. van Valkenburg. *Network Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 2. edition, 1964.
- [Wie93] Remco J. Wiegerink. *Analysis and Synthesis of MOS Translinear Circuits*. Kluwer, Boston/Dordrecht/London, 1993.
- [Win96] Franz Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag, Wien, 1996.
- [Wol99] Stephen Wolfram. *The mathematica book*. Wolfram Media, Champaign, Ill., 4. edition, 1999.

Akademischer Werdegang

Dipl.-Math. David Ilsen,
geboren am 11. Januar 1977 in Köln.

- 06/1996 Abitur, Main-Taunus-Schule, Hofheim a.Ts.
- 10/1996 bis 09/1997 Student der Universität Dortmund
10/1998 bis 09/1999
- 10/1999 Vordiplom in Mathematik, Universität Dortmund
- seit 10/1999 Student der Technischen Universität Kaiserslautern
- 08/2000 bis 12/2000 Gaststudent an der University of California, Berkeley, USA
- 04/2002 Diplom in Mathematik, Technische Universität Kaiserslautern
- 04/2002 bis 03/2005 Stipendiat im DFG-Graduiertenkolleg "Mathematik und Praxis"

Curriculum Vitae

Dipl.-Math. David Ilsen,
born on January 11, 1977, in Cologne, Germany.

- 06/1996 Abitur, Main-Taunus-Schule, Hofheim a.Ts., Germany
- 10/1996 till 09/1997 student at the University of Dortmund, Germany
10/1998 till 09/1999
- 10/1999 Vordiplom (intermediate diploma) in mathematics,
University of Dortmund
- 10/1999 till 05/2006 student at the University of Kaiserslautern, Germany
- 08/2000 till 12/2000 foreign student at the University of California, Berkeley, USA
- 04/2002 Diplom (graduation) in mathematics, University of Kaiserslautern
- 04/2002 till 03/2005 scholarship of the DFG-Graduiertenkolleg "Mathematik und Praxis"