Methodological Issues in Structural and Syntactical Theories

Eugen Ardeleanu
RISC-Linz
MSI-Cornell
ardeleanu@risc.uni-linz.ac.at

Abstract

An outline for the study of the methodology of structural theories emphasising invariant-theoretic methods, and of the syntactical theories emphasising combinatory algebras and completion methods in symbolic computation and artificial intelligence is presented on a level of abstraction which permits a unifying viewpoint on problems in symbolic computation and artificial intelligence. We refer to applications in computational polynomial ideal theory and in general problem-solving and modeling in the sense of AI research.

Key words: completion, complete set of invariants, polynomial ideal theory, AI problem-solving, combinatory algebras, modeling.

1 Introduction

In this article we discuss two methodological principles commonly used in mathematics and computer science.

The first principle we refer to, integrates *structural methods* mainly used to provide for an *external characterization* of relational (algebraic) structures.

The second envisioned principle embodies *syntactical methods* usually in force when dealing with an *internal characterization* of equational theories.

Correspondingly, we will concentrate on two instances of these principles:

1. invariant theories, and
2. theories with completion.

It is our opinion that both areas of symbolic computation and artificial intelligence could benefit from a combined structural-syntactical approach to some of their problems. The recent work of Engeler on combinatory differential

fields and, respectively, combinatory algebras shows that unifying structures serving as a ground field for both theories do exist.

In Section 2. we discuss some aspects of structural theories and their characterizing features: invariants, representations and completeness. We also sketch here two possible applications as a logical framework for symbolic computation and as a tool in the design of symbolic computation systems. A scheme for a new approach to AI problem-solving based on a heuristically guided search for invariants characterizing a given problem is also included.

In Section 3. we concentrate on syntactical theories, in particular the completion method. One of its most pertinent application is construction of standard (Gröbner) bases by Buchberger's completion algorithm in polynomial ideal theory, which, in turn, represents an important topic of computer algebra. We consider general problem-solving (from AI viewpoint) and stress the parallelism with symbolic computation in applying the two methods. An example is also included. Finally we briefly discuss combinatory algebras and application to modeling, knowledge-bases and knowledge representation. Section 4. concludes the article.

2 Structural Theories

2.1 Invariant Theories

The relation between *Invariants, Representation Theorems*, and *Completeness Theorems* appears frequently in algebra and logic.

Let $\mathbf{C}$ be an arbitrary but fixed category of algebraic structures (i.e objects and morphisms obeying the usual axioms). Consider in $\mathbf{C}$ a fixed universal construct $\mathcal{U}$, i.e an object and a set of distinguished morphisms (e.g sums, products), assuming, of course, it exists. Then, one is interested in the following problems formulated on a general level:

- Determine the structure of the objects in $\mathbf{C}$ using $\mathcal{U}$.
- Determine the objects with the simplest structure which are preserved under (a certain class of) morphisms, i.e the invariants.
- Determine the complete set of invariants with which one is capable of characterizing any object in $\mathbf{C}$.

Examples: The theory of modules (with the well known instantiations, e.g the theory of finitely generated abelian groups, the theory of linear transformations for finite dimensional vector spaces), cf. [16].

The theory of quadratic forms, cf. [20].

The algebraic theory of dynamical systems, cf. [14].

The representation theorem for boolean functions, cf. [15].

The concept of indexed systems, e.g logical fiberings, cf. [18], and indexed categories and related topics of increasing interest for the foundations of computer science, logic and mathematics.

2.2 A Logical Frame for Symbolic Computation

The central problem in symbolic computation is the design of representation schemes for more complex, abstract (algebraic) concepts in terms of the one already implemented on the computer (a variant of bootstrapping).

In general, the approach taken for solving this problem consists of representing the "new" concept by means of formulas involving only symbols standing for already implemented concepts (algebraic constructs). The operations on the newly introduced concepts are then represented as operations on the set of definitory formulas (e.g. the introduction of algebraic numbers by means of the corresponding minimal polynomial equation).

We will describe a new (tentative) approach to the development of a method for representing *universal algebraic constructs*.

Let $\mathbf{A} = (A, R, F, C)$ be a relational structure implementable on a computer, i.e. the elements of A are representable as symbols and the relations and functions correspond to operations performable on these symbols. Let $\mathbf{S}$ be a category, i.e. a collection of sets with an algebraic structure and morphisms between them, and $\mathcal{U}$ a universal construct, i.e. a procedure that takes as arguments objects of the category, and produces an universal object. Let $a \in A$ be an arbitrary element.

Definition. We say that $\mathbf{I} = \langle \mathbf{S}, \mathbf{A} \rangle$ is an *invariant theory*, and that $a \in A$ *characterizes* the structure $S_a \in \mathbf{S}$ if and only if, there exist a set Φ_a of *representation formulas* $\{\phi(a, a_1, ..., a_n)\}$ and a corresponding set Ψ_a of *invariant formulas* $\{\psi(a_1, ..., a_n)\}$ such that

$$S_a \cong \mathcal{U}(S_{a_1}, ..., S_{a_n})$$

via an isomorphism in $\mathbf{S}$ for all $(a_1, ..., a_n)$ for which the formulas in Φ_a and Ψ_a hold.

Example. Consider the category of abelian groups $\mathbf{Ab}$ and the relational structure of natural numbers N. For $a \in N$ let Φ_a be the set of all formulas expressing that a is the product of $a_1, ..., a_n$, $a = a_1 \cdots a_n$, and Ψ_a be the set of formulas imposing restrictions on the a_i appearing in Φ_a, $a_{i+1} \equiv 0 \pmod{a_i}$ (specifying the invariant factors). In $\mathbf{Ab}$ we refer to the direct sum $\oplus$ as our universal construct $\mathcal{U}$. Then, the structure theorem of finitely generated abelian groups provides the representation of $\mathcal{U}$.

Let now for all $a \in A$, Φ_a denote the set of representation formulas of a, and Ψ_a denote the set of invariant formulas relative to an invariant theory $\mathbf{I} = \langle \mathbf{S}, \mathbf{A} \rangle$

(given by a structure theorem in $\mathbf{S}$). Consider $\Upsilon_a = \Phi_a \cup \Psi_a$. The formula

$$\bigwedge_{a \in A} \Upsilon_a$$

represents $\mathcal{U}$. (Here $\bigwedge_{a \in A} \Upsilon_a$ is to be regarded as a *symbol* with the intended meaning that all formulas in Υ_a hold for every $a \in A$. We still have to define the operations on such symbols matching the usual properties of universal constructs.)

Invariant theories $\mathbf{I} = \langle \mathbf{S}, \mathbf{A} \rangle$ are then our frame for representing universal constructs in the category $\mathbf{S}$ by means of sets of formulas in the relational structure $\mathbf{A}$.

2.3 AI Problem-Solving

We consider "problem-solving" in the sense used in artificial intelligence, as the discipline concerned with the development of general applicable intelligent strategies which implemented on computers can (help to) provide solutions by imitating human reasoning.

For defining what we mean by a problem we employ a similar to "state-operators" description, cf. [18]. Given a relational structure $\mathbf{A}$, we first define a *state space* having as elements *states*. A state is defined by a finite set of formulas in $\mathbf{A}$; sets of formulas defining distinct states differ only in the values of the constants appearing in the formulas. A *state construction problem* expressed in $\mathbf{A}$ consists of a couple $StConP = (StSp, ConR)$ where $StSp$ represents the state space, and $ConR$ the *set of construction rules*. The states in $StSp$ and the construction rules in $ConR$ are characterized by formulas involving only constants, relations and functions from $\mathbf{A}$. Let $s, s' \in StSp$ and $cr \in ConR$. If rule cr applied to state s transforms it into state s' we say that s' is *reachable* from s. In order to establish an analogy with the reduction technique in symbolic computation described in Section 3., we will adopt the hypothesis that every rule is reversible in the sense that, if s' can be reached from s by cr then s can be reached from s' by applying the corresponding reversed rule r. We say that s' *reduces* to s via r, in symbols $s' \rightarrow_r s$, and call r a *reduction rule*. We will use $RedR$ to denote the set of reduction rules. A state s' can be reached from a state s if there exist a finite set of construction rules $\{cr_i\} \subseteq ConR$ which successively applied to s transforms it into s'. If this is the case, we will say that s' is in the *reachable space* of s, $s' \in ReSp[s]$.

In this context we will not be concerned with finding a solution (i.e a way to transform s in s' by specifying the construction rules $\{cr_i\}$ and the order in which these are applied) but merely to state necessary and sufficient conditions for the existence of such a solution. Thus, we assume that a necessary and sufficient condition for state s' to be reached from state s is that s' can be reduced to s by successively applying reduction rules from $RedR$.

> *"Solving problems by heuristically guided, trial-and-error search in a space of possible solutions is a dominant theme in artificial-intelligence research."* (cf. [18])

We propose a new approach to the subject by considering the study of necessary and sufficient conditions for a problem to be solvable, i.e:

> *Finding necessary and sufficient conditions for the solvability of the problems by heuristically guided, trial-and-error search for* invariants *corresponding to the reduction rules R.*

One could name such an attempt: *Invariant Driven Reasoning.*

The following Definitions 1', 2', 3', 4' and Theorem 1' are analogous to Definitions 1, 2, 3, 4 and Theorem 1 in Section 3.1.

Definition 1'. The *state reachability problem* consists of a triple $StReP = (InSt, R, FinSt)$, where $InSt \subseteq StSp$ is the set of *initial states*, $FinSt \subseteq StSp$ is the set of *final states*, and $R \subseteq RedR$ is the set of admissible reduction rules. (For simplicity reasons we will consider the case in which $InSt = \{is\}$ and $FinS = \{fs\}$ are singletons and $R = ConR \cup RedR$; which corresponds to considering the transitive symmetric closure of the reduction (or construction relation), cf. [23].)

The problem can now be stated as: Decide if the final state fs can be reduced to the initial state is by successively applying reduction rules from R, i.e $fs \in ReSp[is]$.

We will introduce now the basic concept for the proposed approach. A formula ϕ expressed in $\mathbf{A}$ which holds in state s' is said to be an *invariant formula* for the set of reduction rules R if for all $s \in StSp$ and $r \in R$ such that $s' \rightarrow_r s$ the formula ϕ holds also in the state s, i.e $\phi(s) = \phi(s')$, where $\phi(s)$ denotes the *value* of the invariant formula ϕ in state s.

Let $\Phi = \{\phi_i\}$ be a set of invariants for problem $StReP$.

Definition 2'. We say that the set of invariants Φ is *compatible* with the problem $StReP = (is, R, fs)$ if the values of the invariants in the initial state are respectively equal to the values of the invariants in the final state

$$\phi_i(is) = \phi_i(fs) \quad \text{for all} \quad i.$$

Definition 3'. The set Φ is a *complete set of invariants* (or an *invariant system*) for the problem $StReP$ if every invariant formula of the problem $StReP$ can be expressed as a formula involving only invariants from Φ.

Conceivably, the invariant system of a problem is not uniquely defined. Let Φ be an invariant system for the problem $StReP$.

Definition 4'. The state reachability problem $StReP$ is *decidable by reduction* with respect to Φ if and only if it can be formulated in the following way:

> The state reachability problem has a solution if and only if Φ is a compatible invariant system for the problem.

Theorem 1'. The set Φ is a complete set of invariants for the problem $StReP$ if and only if the state reachability problem $StReP$ is decidable by reduction.

Proof. Suppose Φ is complete with respect to R and compatible with respect to $StReP = (is, R, fc)$, and that $StReP$ has no solution. Let $s \in StSp$ be an arbitrary state and consider the formula $\phi(s)$: $\exists R'\ (R' \subseteq R\ finite; fc \rightarrow_{R'} s)$. Then, if $\phi(s)$ holds then $\phi(s')$ also holds for all states s' and rules r such that $s \rightarrow_r s'$, and if $\phi(s)$ does not hold then also $\phi(s')$ does not hold (recall that we have considered $\rightarrow_R$ as the symmetric transitive closure of the reduction relation). This shows that ϕ is an invariant formula for R. But now $\phi(fs)$ holds (trivially) while $\phi(is)$ does not, since $StReP$ has no solution. This is a contradiction with the fact that Φ is a complete, compatible invariant system.

Suppose now that Φ is complete and $StReP$ is solvable. Then clearly Φ is compatible.

For the reverse implication of the theorem suppose $StReP$ is decidable by reduction with respect to R and Φ is not complete. Then we can construct an invariant function ϕ_0 such that $\phi_0(fc) \neq \phi_0(ic)$ (because Φ is not complete) and this would contradict the assumption that $StReP$ is solvable.

Example. Let $\mathbf{A} = N$, the set of natural numbers. Consider a rectangular board with dimensions $a, b \in N$, completely divided into squares of dimension 1. Suppose we have at our disposal at least $\lfloor a \cdot b/2 \rfloor$ domino tiles of dimensions $2, 1$. We can take away squares from the board without changing the position of the remaining squares obtaining in this way different configurations. The problem is:

> Given a certain configuration, decide whether it can be completely covered with non-overlapping domino tiles such that only the squares present in the configuration are covered.

We will first restate the problem as a state construction problem $StConP$. For this, let $StSp$ be the state space corresponding to a given configuration, having as elements (states) different coverings of the given configuration (board) with domino tiles, and $ConR$ the set of construction rules containing just one rule cr: *put a domino tile on two adjacent not covered squares in the configuration* (adjacent means here that they have a common edge).

The state construction problem $StConP$ can now be formulated as:

> Given a board configuration and an initial state (assume, for simplicity, it is the one in which no tile has been placed, thus it actually coincides with the board configuration) and a final state (assume it is the one in which the given configuration has been completely covered with tiles) decide whether, using the construction rule cr, the initial state can be transformed into the final one, and, if so, show how can this be performed.

The reduction rule corresponding to cr is the rule r: *remove a tile from two adjacent squares in the configuration.*

The corresponding state reachability problem $StReP$ can now be formulated as:

> Given a board configuration, an initial and a final state (as in $StConP$) decide whether the final state can be reduced to the initial one.

Consider a coordinate system $N \times N$ with the origin in the lower left corner of the board. Then, each square is represented by a pair of natural numbers $(x, y) \in N \times N$, $0 \leq x \leq a, 0 \leq y \leq b$. For each state $s \in StSp$, i.e partial covering with tiles of the initial configuration, let us define the following predicates:

$cover(x, y) = true$ if and only if the square (x, y) is covered by a tile in state s; $cover$ characterizes the state;

$sing(x, y) = true$ if and only if the square (x, y) is in a *singular position* in state s, i.e all adjacent squares are covered with tiles;

$even(x, y) = true$ if and only if $x + y \equiv 0 \pmod 2$.

Let us denote by $N_{cover}, N_{sing}, N_{even}$ the number of positions (x, y) for which the corresponding predicates take the value $true$ (in state s). These all are functions defined on $StSp$ with values in N.

For the initial state is, let us define the predicate:

$in - config(x, y) = true$ if and only if the square at position (x, y) is in the board configuration; thus $in - config$ characterizes the configuration of the board and, since we have considered by hypothesis that the board configuration coincides with the initial state, it also characterizes the initial state. Clearly, $N_{in-config}$ represents then the number of positions (squares) in the initial state (board configuration). The solvability of $StReP$ is equivalent to the conditions: $N_{cover}(is) = 0$ and $N_{cover}(fs) = N_{in-config}$.

Let us now put into evidence some of the invariants. It is easy to see that $N_d = N_{even} - N_{\neg even}$ is an invariant formula for the reduction rule r because a tile can be removed only from two adjacent squares. For the same reason N_{sing} is also an invariant. Clearly these two functions are independent, in the sense that they can take any value (in their codomains) independently of one another. It is also clear that N_0 in $N_{cov} \equiv N_0 \pmod 2$ is an invariant, but, since $N_{cover} = N_{even} + N_{\neg even} = 2N_{even} - N_d \equiv N_d \pmod 2$, it is not an independent (invariant) function. Now, since $N_{cover}(is) = 0 \equiv 0 \pmod 2$

it follows $N_0 = 0$, hence $N_{in-config} \equiv 0 \pmod 2$ is a necessary condition. Further, since $N_d(fs) = 0$ and $N_{sing}(fs) = 0$ it follows that, for the set of invariants $\{N_d, N_{sing}\}$ to be compatible, $N_d(is) = N_{sing}(is) = 0$. However the set $\{N_d, N_{sing}\}$ is not a complete invariant set. We were not able to find such a set for the problem. However, let us notice the importance the "right" data structure has. Indeed, expressed in graph-theoretic terms, the problems becomes the well studied *perfect matching in a bipartite graph*; solved in terms of graph invariants! (cf. [24]). We should also point out here that, in general, we cannot reasonably hope for the existence of an algorithm which, performing on $StRePs$ as input, will give an invariant system as output (even in the simple problem considered in the example, the relational structure we had to consider for expressing the invariants, contains as subsystem the number-theoretic formal system where incompleteness theorems hold, cf. [15]). This situation contrasts with the situation we will have when considering Buchberger's completion algorithm for polynomials in Section 3.1. However, it is not difficult to construct simple $StRePs$ where a complete set of invariants can be exhibited.

3 The Methodology of Syntactical Theories

3.1 Completion and Computer Algebra

It is often the case in computer science that the *syntactical* approach, is emphasized, cf. [13], mainly because of immediate possibilities for applications and implementations, e.g equational theories (varieties in universal algebras defined by a set of equations) in algebraic specification of programming languages, rewriting systems (with instantiations as grammars, Thue systems, etc), unification (with implementations as Prolog).

The main problems in syntactical theories are not to determine the structure of the objects, but to develop (constructive) methods for "simplifying" them. The process is known as *reduction to normal forms* and the *termination* and *uniqueness* of the result of reduction are the crucial points.

To prove termination the common technique consists of defining a total well ordering on the set of objects compatible with the reduction relation between the "more" and "less" complex objects, cf. [7].

To prove uniqueness comes down to prove that the reduction relation has (some variant of) the Church-Rosser property, which, roughly, ensures that the reduction to normal forms does not dependent on the different ways it can be performed, i.e normal forms are unique, cf. [7].

There have been also attempts to introduce a categorical viewpoint in dealing with syntactical theories; the objects are the well-formed expressions in the theory and the morphisms are the reduction rules. In this way some fundamental properties of syntactical theories correspond to universal categorical constructs, cf. [11].

The general method of designing algorithms by completion (for a survey see [3]) is of ancient vintage in algebra. It is used nowadays in symbolic computation as a basic tool. The most famous instance is Buchberger's algorithm for "completion" to standard (Gröbner) bases in polynomial ideal theory. We will first briefly describe the problem and the idea of the method in this context (for the exact definitions and proofs see [4]).

Definition 1. The *ideal membership problem*:
Given a polynomial, say $f \in R[x_1, ..., x_n]$ and a finite set of polynomials $S = \{f_1, ..., f_m\} \subset R[x_1, ..., x_m]$, decide whether f belongs to the ideal generated by S, $f \in I[S]$.

The natural idea to solve this problem is to start to divide f by polynomials f_i, $f = p_i f_i + r$ (we say that f *reduces* to r by f_i, in symbols $f \rightarrow_{f_i} r$) and continue like in Euclid's algorithm (in the case of univariate polynomials it *is* the Euclidean algorithm), with r as f. We can view this as a reduction process and we have to ensure termination and uniqueness. Termination is ensured by introducing a total ordering $<$ on $\mathbf{R}[x_1, ..., x_n]$, (e.g a graded lexicographical ordering) compatible with the reduction relation. Thus, if f reduces to r, then $r < f$. It turns out that this ordering is noetherian, by Hilbert's basis theorem.

Definition 2. We say that f *reduces to* f' *with respect to* S, $f \rightarrow_S f'$ if and only if there exist $f_i, ..., f_j \in S$ such that

$$f \rightarrow_{f_i} ... \rightarrow_{f_j} f'.$$

It is clear that, if $f \rightarrow_S 0$, then $f \in I[S]$. However, the converse is in general not true. For the converse to hold, we have to "complete" the set S to a so-called *standard (Gröbner) base*, first introduced by Buchberger, cf. [4]. If no polynomial in this set can be reduced via some other polynomial in the set, we have a *reduced* standard base. Motivated by the following Theorem 1., we will take the liberty of introducing some new terminology: we will call standard (Gröbner) bases *complete generating sets* and reduced standard (Gröbner) bases, *complete bases*.

There are many (syntactical) characterizations for the complete sets, cf. [23]; among others, the one stating that S is a complete set if and only if every polynomial f reduces to a unique normal form (i.e a polynomial f' which cannot be further reduced) with respect to S.

Definition 3. A set of polynomials S is a *complete set* if one of the following equivalent conditions is fulfilled:
The relation $\rightarrow_S$ has the Church-Rosser property.
For all $f_i, f_j \in S$, $spoly(f_i, f_j) \rightarrow_S 0$.

(Here $spoly(f_i, f_j)$ is the *substraction polynomial* of f_i and f_j defined as the difference between the polynomials obtained by reducing the least common multiple of the leading power products of the two polynomials with respect to f_i and f_j.) The equivalence of the conditions in Definition 3. is known as the Buchberger's theorem. This result makes it possible to design an algorithm which accepts as input S' a finite set of polynomials and produces as output a complete set S such that $I[S] = I[S']$, cf. [4]. Hence the ideal membership problem is decidable.

We can now restate the ideal membership problem (cf. Definition 1.) as:

Definition 4. The ideal membership problem is *decidable by reduction* with respect to S if and only if it can be formulated in the following way:

$$f \in I[S] \quad \text{if and only if} \quad f \rightarrow_S 0.$$

We note that if $f \rightarrow_S f'$ then the coset $\overline{f}_{I[S]}$ of f modulo $I[S]$ is an *invariant* with respect to the reduction relation $\rightarrow_S$, i.e

$$\overline{f}_{I[S]} = \overline{f'}_{I[S]}.$$

The following theorem states a necessary and sufficient condition for the solvability of the ideal membership problem.

Theorem 1. The set S is a complete set for $I[S]$ if and only if the ideal membership problem is decidable by reduction with respect to S.

Proof. Cf. [4].

3.2 Combinatory Algebras

First let us recall some basic notions from [8] and [10]. Starting with a vocabulary of basic facts, let this be FT - the set of functionality types, a cumulative hierarchy of sets of expressions $S_n(FT)$ is built by

$$S_0 = FT$$

$$S_{n+1}(FT) = S_n(FT) \cup \{\sigma \rightarrow s : \sigma \subseteq S_n(FT),\ \sigma\ finite,\ s \in S_n(FT)\}$$

The language for representing the knowledge (about the functionality types as states of activities) is then

$$LS = \bigcup_n S_n(FT)$$

and the objects which we choose as *states representatives* are the subsets of LS. The application of one object (in our interpretation the set of states of an activity) S_A to another object S_B is defined as

$$S_A \cdot S_B = \{s : \exists \sigma \subseteq S_B . \sigma \rightarrow s \in S_A\}$$

The set S_{FT} of all subsets of LS with the binary operation $\cdot$ as defined above, becomes an algebraic structure, a so called *combinatory algebra* $\mathbf{S}_{FT} = \langle S_{FT}, \cdot \rangle$.

3.3 Modeling

The name *activity* has already appeared many times in artificial intelligence in connection with *planning* and *scheduling*, cf. [5], [22], as well as in operational research in connection with *evaluation* and *optimization techniques*, cf. [1]. In all the contexts in which the denotation "activity" is used, the common interpretation is to designate a basic entity with which, by some sort of assembly process, other activities can be built up in order to *model* and/or *represent* a certain *system*.

In our model, based on an augmented variant of the model described in [10] and [21], an activity has input/output connections and states, and is to be regarded as a cognitive tool to recognize the structure of a data (entity) in a database. The states provide for an internal characterization of the functionality-type of an activity, as in basic algebraic system theory, cf. [14], and are to be regarded as elements of a combinatory algebra which is the mathematical environment suited to treat local (by connectivity-type parameters) as well as global aspects of the data (by functionality-type). The coupling of two activities is based on input/ output type matching and this is used to direct the search procedure. The concept of activity as defined in this article is a dynamical one supporting model-driven programming.

We present now an example of application on a problem originating in robotics and presently dealt with in model-driven programming.

The following problem is taken from [12]:

- Write a computer program for a robot to automatically assemble a complex piece, e.g a steering column, out of simple (primitive) components (pieces).

Formulated in model-driven programming terminology this comes down to:

- Compile an exploded diagram from a database of part descriptions and simple assembly tasks into the code needed to drive the robot.

We will now try to point out the relevant aspects of such an enterprise.

1. Each piece can be assembled with several other pieces at fixed connection points. (We will speak about connection points even if surfaces or volumes

are actually involved in the assembly process.) At each connection point only one type of connection is possible (e.g. screwing, insertion, lining) and the connection types of the two pieces assembled at a certain connection point should match.

2. Each piece can serve to several functionalities (e.g the links of a robot can have a rotatorial, translatorial or combined movement, cf. [6]) and the set of functionality types associated to an assembly is an element of a cumulative hierarchy of functionality types.

3. Situations may occur in which the order in which the assembly process is performed is of relevance, i.e assembling in certain (wrong) order may prevent the completion of the process.

We will now generalize the considerations from the previous example, give the formal definitions and come up with the model. Finally we will show how this model offers a reasonable solution to the problem mentioned above.

In our model the set of states S_A of an arbitrary activity A is simply an element of $\mathbf{S}_{FT}$ and the composition of two activities is modeled by the binary operation $\cdot$. This is a non-commutative and non-associative operation.

Consider now a set CT whose elements are called *connectivity types* and for each connectivity type $c \in CT$ a *set of parameters* $P_c = \{p_1, ..., p_{n_c}\}$. Consider also a set S whose elements are called *states*. Identifying $S = S_{FT}$ we are now in the position to give the formal definition for the concept of activity.

An *activity* A consists of a couple (C_A, S_A) where C_A is the *set of connections* and $S_A \in S_{FT}$ is the *set of states* (or the *generalized functionality type*) of the activity A.

The aspects mentioned above are formalized as follows:

1. Let A be an activity. Then to each connection $a_i \in C_A$ there is associated a connectivity type $ct(a_i) \in CT$ and a *set of values* $V_{P_{ct(a_i)}} = \{v^1, ..., v^{n_{ct(a_i)}}\}$ for parameters $P_{ct(a_i)}$ uniquely determining the particular connection within the set of connections having the same connectivity type. Thus, for an arbitrary connection, the corresponding connectivity type and set of parameters represent, respectively, a qualitative and a quantitative identification.

2. Activities are the model for the components (pieces) in the example in Section 2. The states of an activity as elements of the combinatory algebra $\mathbf{S}_{FT}$ can be viewed as generalized functionality types and model the functionality type of the corresponding component.

3. The binary operation $\cdot$ in $\mathbf{S}_{FT}$ is non-associative, corresponding to the fact that the order in which the components are assembled is relevant. Thus, we can speak of a partial order relation defined on the set of activities,

relative to a (complex) component; it corresponds to the order in which the simple components (pieces) are assembled.

The assembly process on components is modelled by the composition of activities via the operation in $\mathbf{S}_{FT}$ under the connectivity type-matching constraint.

Let $A = (C_A, S_A)$ and $B = (C_B, S_B)$ be two activities, $C_A = \{a_1, ..., a_{n_A}\}$, $C_B = \{b_1, ..., b_{n_B}\}$, $n_A, n_B \in \mathbf{N}$ the number of connections of activity A and B, $1 \leq i_1 \leq ... \leq i_m \leq n_A$, $1 \leq j_1 \leq ... \leq j_m \leq n_B$ and $ct(a_{i_k}), ct(b_{j_k}) \in CT$ for all $k = 1, ..., m$, the connectivity types. Then the necessary conditions for the composition of activities A and B at the connections $\{a_{i_1}, ..., a_{i_m}\}$ and $\{b_{j_1}, ..., b_{j_m}\}$ are:

$$ct(a_{i_k}) = ct(b_{j_k})$$

$$v^l_{a_{i_k}} = v^l_{b_{j_k}}$$

$$n_{ct(a_{i_k})} = n_{ct(b_{j_k})} = n_k$$

for all $l = 1, ..., n_k$ and $k = 1, ..., m$.

We assume that at each connection of any activity at most one activity can be connected. In this way the restrictions are put into evidence by considering the connectivity types; a particular connectivity type at a particular connectivity point usually renders inconsistent some functionality types

The *composed activity* $(AB) = (C_{AB}, S_{AB})$ is defined by:

$$C_{AB} = C_A \cup C_B - \{a_{i_1}, ..., a_{i_m}\} - \{b_{j_1}, ..., b_{j_m}\}$$

$$S_{AB} = S_A \cdot S_B$$

For example, let $\{r(otation), t(ranslation), s(crew)\}$ be a set of functionality types. Then an activity with functionality type $\{r, t\} \rightarrow s$ has the potential that, assembled with an activity with functionality type r or t results in an activity with functionality type s. Further, such one can be obtained by composing an activity with functionality type $\{s \rightarrow (\{r, t\} \rightarrow s)\}$ with an activity having functionality type $\{s, ...\}$, etc. A set of interconnected activities (according to the rules described above) forms an *activity graph*.

Activity graphs are our models for complex components built up from simple ones. The interesting case is, of course, regarding *connex* graphs, for we can always restrict ourselves to connex components and treat them separately. The construction of an activity graph can now be represented by an expression in $\mathbf{S}_{FT}$, e.g the expression $A((BC)D)(EF)$, involving the activities A, B, C, D, E, and F, stands for the composition process by which first B is connected with C, then the resulting activity with D, then A with the previous resulted activity, etc. The convention of associativity is on the left.

In this way the result of building an activity graph G from activities $A_1, ..., A_n$ can also be viewed as an activity $G = (C_G, S_G)$ The problem stated at the beginning of Section 2 can now be formulated as:

- Given a (generalized) connectivity type C_0 and a (generalized) functionality type F_0, construct an activity graph G from a database of activities involving activities $A_1, ..., A_n$ such that the equation

$$F_0 = (...((S_G \cdot S_{A_1}) \cdot S_{A_2} \cdots S_{A_n}) \tag{1}$$

holds in the algebra $\mathbf{S}_{FT}$ and the connectivity type constraints for activities composition are fulfilled.

We note here that the notions "parts descriptions" and "assembly tasks" in the original formulation of the problem were substituted by a single concept of activity and the compilation requirement by the construction of an activity graph, which can be, in principle, effectively computed.

The question now naturally arises, which composition processes can be represented by activity graphs, i.e under what conditions is Eq. (1) solvable for S_G. The answer is known from combinatory algebra (cf. [8]): it is the case if and only if the functional dependence:

$$F_0 = f(S_{A_1}, ..., S_{A_n})$$

is continuous in the following sense:

$$f(S_{A_1}, ..., S_{A_n}) = \bigcup_{\sigma_i \subseteq S_{A_i}} f(\sigma_1, ..., \sigma_n), \quad \sigma_i \ finite$$

and then f can be represented by S_G as:

$$S_G = \{(\sigma_1 \to (\sigma_2 \to ...(\sigma_n \to s)...) : a \in f(\sigma_1, ..., \sigma_n),\ \sigma_i \subseteq S_{A_i},\ \sigma_i \ finite\}$$

.

The main advantage of this approach is that the problem stated in Section 2., which appears frequently in database theory, can now be stated as an equation in the corresponding combinatory algebra of functionality types with restrictions imposed by connectivity type matching. It is the language of constraint logic programming that successfully deals with such problems, in which the result of each query has to be checked against consistency constraints.

Comparing the concepts of activity and activity graphs as defined in this article with similar concepts in the literature we note that our approach supports type-matching by means of connectivity types (cf. [5]), quantitative analysis and optimization by means of parameter sets (cf. [1]), and equation solving driven inference mechanisms by means of (generalized) functionality types (cf. [8] [10]). These concepts can be used as classification and comparison techniques (cf. [17]) as well as dynamic problem solving tools.

4 Conclusions and Further Research

We discussed some methodological aspects of structural and syntactical theories. We have presented the basic idea of a new approach to problem solving from a certain unifying general perspective as invariant driven reasoning. While there is no way to systematically search for invariants, (i.e computer programs that "produce" such invariant formulas) we can envision programs that could test for invariant properties. In general, no completion algorithm exist, but since we know examples (in symbolic computation) were such algorithms do exist, it seems worthy to undertake such a research also in other particular fields.

An attempt for describing Robinson's resolution procedure as an instance (model) of the completion principle was made in [3]. It would be of interest to restate the resolution method using the notions introduced in this article.

Another research direction could be, given a relational structure **A** in which we express a problem $StReP$, to embed **A** into a richer algebraic structure where the invariants are solutions (fixed points) of certain equations (operators). Such a program was realized by Engeler [9], by constructing a combinatory model for programmable functions defined on a relational structure.

We have introduced the concept of activity and activity graphs for modelling complex assembly processes. The construction of an activity graph as a model for a given process is driven by solving an equation in the attached combinatory algebra of generalized functionality types and subject to connectivity type matching constraints.

Even if the construction of an activity graph is effectively computable, the actual implementation seems to be a difficult task. We believe that Prolog, or a Prolog-like programming language with its powerful backtracking and unification/resolution algorithms, could be suited for an implementation.

Another direction for further research could be the investigation of applications in parallelism (computation, manipulation).

Finally we think of applying optimization techniques in the process of constructing the activity graphs.

References

[1] Ackoff, L., Sasieni, W. (1970). *Fundamentals of Operational Research*, Addison-Wesley.

[2] Ardeleanu, E. (1992). Completion And Invariant Theory in Symbolic Computation And Artificial Intelligence, *Conference on Artificial Intelligence and Symbolic Mathematical Computation*, Calmet, J., ed., Springer LNCS (to appear).

[3] Buchberger, B., (1987). History and Basic Features of the Critical-pair/completion Procedure, *J. Symbolic Computation*, **3**.

[4] Buchberger, B., (1976). A Theoretical Basis for the Reduction of Polynomials to Canonical Form, *ACM SIGSAM Bull.*, **10**.

[5] Cohen, P. R., Feigenbaum, E. A., eds., (1982). *The Handbook of Artificial Intelligence*, vol. **3**, Pitman.

[6] Craig, J. J., (1986). *Introduction to Robotics*, Addison-Wesley.

[7] Dershowitz, N., Jouannaud, J.-P. (1990). Rewrite Systems, *Handbook of Theoretical Computer Science*, North-Holland.

[8] Engeler, E., (1986). Cumulative Logic Programs and Modelling, *Logic Colloquium '86*, Drake, F. R., Truss, J. K., eds., Elsevier.

[9] Engeler, E., (1990). Combinatory Differential Fields, *Theoretical Computer Science* **72**.

[10] Engeler, E., (1992). Sketch of a New Discipline of Modelling, *J. Symbolic Computation* (to appear).

[11] Goguen, J., (1986). What Is Unification?, *Categories in Computer Science*, Springer.

[12] Hopcroft, J. E., (1986). The Impact of Robotics on Computer Science, *Communications of the ACM*, **29**.

[13] Jouannaud, J-P., (1990). Syntactic Theories, *Mathematical Foundations of Computer Science*, Rovan, B., ed., LNCS **452**, Springer.

[14] Kalman, R., Arbib, M., Falb, P., (1965). *Topics in Mathematical System Theory*, McGraw-Hill.

[15] Kleene, S., (1967). *Introduction to Metamathematics*, North-Holland.

[16] Lang, S., (1984). *Algebra*, Addison-Wesley.

[17] Lockermann, P. C., Moerkotte, G., (1991). On the Notion of Concept, *10-th Int. Conf. Entity-Relationship Approach*, Teory, T. J., ed.

[18] Nilsson, N., (1971). *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill.

[18] Pfalzgraf, J., (1991). Logical Fiberings and Polycontextural Systems, *Fundamentals of Artificial Intelligence Research*, Jorrand, Ph., Kelemen, J., eds., LNCS **535**, Springer.

[20] Rota G.-C., Sturmfels B., (1988). Introduction to Invariant Theory in Superalgebras, *Invariant Theory and Tableaux*, Springer.

[21] Schwärzler, G. (1992). An Algebraic Approach to Knowledge-Based Modeling, *Conference on Artificial Intelligence and Symbolic Mathematical Computation*, Calmet, J., ed., Springer, LNCS (to appear).

[22] Vere, S. A., (1983). Planning in Time: Windows and Durations for Activities and Goals, *IEEE Trans. Pattern Anal. Mach. Intell.*, **PAMI-5**.

[23] Winkler, F., (1984). The Church-Rosser Property in Computer Algebra and Special Theorem Proving, *PhD Thesis*, University of Linz.

[24] Zykov, A. A., (1990). *Fundamentals of Graph Theory*, BCS Associates.