A Categorical Critical-pair Completion Algorithm

KAREL STOKKERMANS

Institut für Computerwissenschaften, University of Salzburg, Austria

We introduce a general critical-pair/completion algorithm, formulated in the language of category theory. It encompasses the Knuth–Bendix procedure for term rewriting systems (also modulo equivalence relations), the Gröbner basis algorithm for polynomial ideal theory, and the resolution procedure for automated theorem proving. We show how these three procedures fit in the general algorithm, and how our approach relates to other categorical modeling approaches to these algorithms, especially term rewriting.

1. Introduction

The problem of finding a general formulation for all so-called critical-pair/completion (CPC) algorithms was posed by Buchberger (1987). In that paper, it is argued that although some unifying approaches for e.g. Knuth–Bendix completion and Gröbner basis construction exist, no satisfactory axiomatic approach to CPC algorithms had been achieved. Buchberger suggests to try an approach based on the notions of pattern, multiplier, and replacement, not for obtaining fast and efficient algorithms (which can never be the aim of a general approach) but out of structural interest. We suggested a possible categorical formulation of some of these notions in Stokkermans (1992) and finally succeeded in presenting a general CPC algorithm in Stokkermans (1995a), formulated in the language of category theory, which encompasses the Gröbner basis algorithm, the Knuth–Bendix procedure (also modulo equivalence relations), and resolution (also for many-valued logics).

The language of categories was chosen as it has proved itself over the last decades to be an effective tool for clarifying the relations between mathematical notions from sometimes widely varying fields—as in our case, where we have to deal with polynomial ideal theory, term algebras, and first-order logic.

It should be stressed that in spite of the not completely unjustified characterization of category theory as “abstract nonsense”, it is by no means straightforward to find a good categorical model for these procedures, as may be seen from the discussion in Section 3. The challenge, after all, is not so much in providing a model, but in finding a model that is sufficiently general yet fits the original procedures as closely as possible, so that one gains insight into the resemblances and especially the subtle differences between the individual algorithms.

E-mail: stokkerm@cosy.sbg.ac.at
In this paper, we present the categorical CPC algorithm, and explain how to fit the three mentioned procedures into it.

Basic knowledge on reduction systems, term rewriting and resolution is assumed. A short discussion on other approaches to relate CPC procedures is provided in Section 2, and an overview of other categorical models of term rewriting (in particular) or reduction systems is given in Section 3. Section 4 explains some special categorical constructions which we used. Section 5 describes our categorical model for CPC, proves the relevant, categorical version of the critical-pair lemma, and outlines the categorical CPC algorithm. The following sections specify the categories in the model for each of the main CPC algorithms: the Knuth–Bendix algorithm in Section 6, the Gröbner basis algorithm in Section 7, and (many-valued) resolution in Section 8.

Finally, in Section 9, a short overview of the connections between previous categorical approaches to rewriting and our categorical model for critical-pair/completion as presented here is given.

2. Historical Overview—Relating CPC Algorithms

An extensive overview of attempts to relate CPC algorithms until the early eighties is given by Buchberger (1987). Here, we only summarize the most relevant remarks from that paper, and then proceed with an overview of the work done in this area since (short of the categorical approaches described in the next section). Most work has been devoted to attempts to view the Gröbner basis algorithm as an instance of the Knuth–Bendix algorithm (modulo associativity and commutativity equivalences), so we first focus on that. We then briefly present the term rewriting approach to resolution-based theorem proving.

In order to view the Gröbner basis algorithm in a rewriting context, one needs to be able to deal with the associativity and commutativity of the arithmetic operators. Therefore, the Peterson–Stickel algorithm presented by Peterson and Stickel (1981), which extended the Knuth–Bendix procedure to the associative-commutative case, can be seen as the first step towards a common treatment. That algorithm has the drawback that it may add new rules to an already completed set of rules (that is, if the set of rules already defines a Church–Rosser relation). Together with Huet (1980), which presented a general, axiomatic treatment for proving confluence of abstract reduction relations and showed that many closure conditions verifying confluence for TRS can be specialized to conditions on critical pairs, this work opened the possibility of treating confluence and critical pair problems in an abstract, domain-independent framework.

Buchberger (1983a) extends the CPC approach used for polynomial rings in the original Gröbner basis algorithm (Buchberger, 1965) to general rings. In this general case, there is no natural decomposition of the generators of an ideal into a “head” and a “rest”. This causes the need of formulating the basic notions of “reduction” and “critical pair” in a new way independent of any “rewrite” nature of the generators.

The paper also gives a set of reduction axioms by which the correctness of the algorithm can be proved and which are preserved when passing from a ring \( R \) to the polynomial ring \( R[x_1, \ldots, x_n] \).


Winkler (1984) gives an introduction to both the Gröbner basis and Knuth–Bendix algorithm and investigates the connection between the Gröbner basis algorithm for poly-
nomial rings and the Peterson–Stickel version of Knuth–Bendix completion. It also discusses the gaps left open in the approaches towards incorporating Buchberger’s Gröbner basis algorithm as a special case of the Knuth–Bendix completion procedure for the associative and commutative case, as in Kandri-Rody and Kapur (1983) and Kandri-Rody and Kapur (1984).

Kandri-Rody et al. (1989) shows how the Gröbner basis algorithm and the Knuth–Bendix procedure can be seen as special cases of a more general completion procedure by the introduction of an extra simplification relation $\Rightarrow$, which turns out to be the identity for Knuth–Bendix completion and simulates the operations on constants of the coefficient field $K$ in the Gröbner basis case.

Pottier (1989) proposes an inference system applicable both to terms and polynomials, based on a well-ordering of critical pairs, and then considers a structure of binary relations and ordered sets in which the common notions of various completion algorithms (for the cases of string rewriting, term rewriting and Gröbner basis computation) are defined. Moreover, investigations towards the generalization of terms in an equational theory and logical formulae are made, in an attempt to automatically establish heuristics for the efficient computation of Gröbner bases, following, among others, Kapur and Narendran (1985).

Bündgen (1990) presents a canonical term rewriting system that specifies multivariate polynomial rings over commutative rings with unity. Then, canonical representations for a set of polynomial equations are computed using the specification found by the term rewriting system and term completion modulo AC-theories as in Peterson and Stickel (1981). In the resulting confluent term rewriting system several rules belong to a single polynomial, thus defining the reduction relation associated with each polynomial. This can be interpreted as ideal completion in $\mathbb{Z}[x_1, \ldots, x_r]$. Then, all rule patterns that can occur in the resulting system are analysed and the different steps of the term completion are related to the corresponding steps in Buchberger’s algorithm.

In Bündgen (1991a), the same work is done for Buchberger’s algorithm for polynomials over finite fields.

These results are subsumed by Bündgen (1991b). There, a uniform presentation of completion in the domains of first order terms, groups, and polynomials is given. This implies that group completion and polynomial completion can be simulated by Knuth–Bendix term completion (modulo AC theories). For polynomial rings over several coefficient domains ($\mathbb{Z}$, $\mathbb{Z}_m$, $GF(q)$, and $\mathbb{Q}$) Gröbner basis construction can be simulated by many-sorted term completion modulo AC. The author presents an algorithm to analyse the set of ground terms which are irreducible with respect to a given term rewriting system. It handles certain well-behaved non-left-linear rules, and AC-operators under some additional restrictions. By means of the characterization obtained, algebraic structures isomorphic to the initial model of a canonical term rewriting system can be analysed. This makes it possible to decide the order of all finitely presented groups for which a canonical term rewriting system exists.

A critical pair transformation procedure is given which can be seen as either a completion strategy or a critical pair criterion. Critical pair transformations are useful during completion of groups and domains containing groups as substructures.

As an infinite set of rewrite rules is needed for the presentation of $\mathbb{Q}$ (in order to ensure that the initial model of the rules in the coefficient sort is isomorphic to the coefficient domain), an infinite term rewriting system is necessary for the simulation.
Another approach is suggested by Marché (1994, 1998), in which normalized rewriting and normalized completion are introduced; the associated completion algorithm generalizes both the Knuth–Bendix procedure (modulo associativity and commutativity) and the Gröbner basis construction. Given a convergent rewrite system $S$ (modulo associativity and commutativity), $S$-normalized rewriting is the relation obtained from constructing the normal form of any left-hand side with respect to $S$ and then rewriting with the rule system to be completed.

A connection between deduction problems in (many-valued) logics and the Gröbner basis algorithm is given by Chazarain et al. (1991), which generalizes methods of classical logic based on the Stone isomorphism between Boolean algebras and Boolean rings. The main result of Chazarain et al. (1991) is a theorem transforming a deduction problem in a many-valued logic to an equivalent problem on ideal membership in a polynomial ring, which can of course be solved by the Gröbner basis algorithm.

In a similar vein, Tătar (1993) proved that the extension of the Gröbner basis algorithm to general reduction rings developed by Buchberger (1983a) can be applied to the Boolean polynomials generated by a set of clauses, and expresses this result in a purely logical framework.

There is also a considerable amount of work done in relating term rewriting techniques to resolution-based theorem proving. A good overview of the different methods developed can be found in Hsiang et al. (1992); an earlier overview can be found in Avenhaus and Madlener (1990). Here, we will only give a brief overview of the most pertinent general developments.

The fundamental idea behind the term rewriting approach to equational theorem proving based on resolution is to treat Boolean formulae as Boolean rewrite rules, and then to apply suitable superposition inferences, to produce new rules. By means of certain reduction inferences the Boolean terms are then simplified using the Boolean rules discovered, and the process is continued until the contradictory rule 1 → 0 (truth implies falsity) is generated.

First approaches based on this idea were studied by Dershowitz (1983) and Hsiang and Dershowitz (1985); they were followed by Kapur and Narendran (1985), Paul (1985), Hsiang (1987), Pottier (1989) (see above), Tătar (1991, 1993), and others.

In Dershowitz (1983), the extension of the Knuth–Bendix completion procedure to associative and commutative functions is used as a semi-decision procedure for equational theories, and applied to diverse problems, among others resolution-like theorem proving in first-order predicate calculus. The problems that may occur regarding termination and abortion with failure are left open.

In Hsiang and Dershowitz (1983) it was shown that for theorem proving purposes in first-order predicate logic, only certain unifications (namely those between terms that are conjunctions of atomic formulae) have to be considered. Calling such terms $N$-terms, a refutational strategy, the $N$-strategy, was developed based on that observation. That term rewriting approach was extended by Hsiang (1987) to first-order predicate calculus with equality, implying that the term rewriting method for clausal theorem proving can be made as powerful as paramodulation and resolution together.

Another extension of the approach by Hsiang and Dershowitz (1983) is accomplished by Paul (1985), where the method is generalized to satisfiable theories; it is shown that the concept of confluent rewriting systems can be extended to any quantifier-free first-order theory, and that rewrite methods can be used even if formulae are kept in clausal form.
3. Historical Overview—Categorical Approaches to Rewriting

Most work in the direction of categorical modeling of CPC algorithms has concentrated on rewriting systems. The main motivation behind this research may have been connected with the successful attempts to relate the theory of lambda calculus to certain categories (namely cartesian closed ones; for more on this we refer to Lambek and Scott, 1986). The step to trying similar approaches to term rewriting systems is then a relatively small one.

However, the first attempt to model rewriting by categorical means predates this. The purpose of Benson (1975) was to establish a proper algebraic framework for further studies of the syntax and parsing of languages and the general theory of translation, compilation, and interpretation. For this, Benson only required a categorical model for string rewriting systems (SRS). For a given SRS, a derivation category $D$ is defined which models arbitrary derivations between strings. These derivation categories are free strict monoidal categories, simple forms of 2-categories. By allowing interchange of consecutive rewrites that apply to disjoint substrings one obtains a quotient category $S$, the syntax category. Two endofunctors are defined on this category modeling concatenation of substrings on either side of a derivation. Then, Benson (1975) forms a category of interchange operators and proves a uniform representability result for SRS.

Huet (1986) describes a more general model. A derivation category for a term rewriting system (TRS) is constructed with as objects terms and as arrows sequences of 1-step reductions on disjoint subterms. The model is then restricted to regular TRS. For those, a so-called computation category is defined: the derivation category quotiented by permutation equivalences—essentially the same transition as from derivation to syntax category by Benson (1975). The objects of this computation category are the terms, and the arrows are permutation classes of parallel derivations from domain to codomain. This computation category has all pushouts, thanks to the fact that confluence is guaranteed (by the absence of critical pairs in regular TRS). This is related to the approach presented in this paper in that we will use certain pushouts to check confluence.

Apparently independent of Benson (1975), Johnson introduces a 2-categorical model for string rewriting (cf. Johnson, 1988, 1998, 1991). His freely constructed 2-category $C_{\Sigma, P}$ (with $\Sigma$ a signature and $P$ a SRS) relates to Benson’s model in the sense that the syntax category $S$ by Benson (1975) is exactly the hom category $C_{\Sigma, P}(\bullet, \bullet)$, with $\bullet$ the unique 0-cell. Johnson generalizes the model to TRS by introducing 3-categories, in which the 1-cells model types, the 2-cells terms, and the 3-cells rewrites. The substitutions (2-cells) are kept separate from the rewrites (3-cells), but the model cannot handle repeated variables, so it only applies to linear rewrites.

Another approach is that formulated by Rydeheard and Stell (1987). They start from a Kleisli category modeling term substitution. The rewrite rules induce a 2-categorical structure on top of this. Separation of substitution and rewrites is achieved by tying the rewrite rules explicitly to the variables used in expressing them. In this model, most general unifiers are (weak) coequalizers. Based on their treatment of equational proofs and unification algorithms, Reichel (1990) gives a 2-categorical formulation of the critical pair situation, namely narrowing a 2-cell against a 1-cell.

However, completion by critical pairs itself, our main algorithmical interest here, is not modeled in any of the above approaches.

In the context of theorem proving, Bonacina and Hsiang (1991) apply category theory, formalizing inference rules (among which resolution) of a given arity $n$ as natural transformations between functors from a category of signatures to the corresponding category.
of sets of $n$-tuples of sentences over that signature. The source functor simply builds the set of relevant tuples, the second splits them in a pair of sets, namely those sentences to be deleted and those to be added on the application of an inference rule.

Based on this framework, Bonacina and Hsiang (1991) continue to describe search plans by equipping the set of proofs over a theory with a proof ordering and using a functorial construction to find minimal proofs. (In addition to proof reduction, monotonicity and relevance of proving strategies are formulated as functors.) While this framework captures one of the components of critical-pair completion (namely completion of sets with respect to a given reduction relation on them), it does not address critical pairs, and the concept does not seem to have a straightforward counterpart in the model; instead the approach focuses on strategies of completing in a far more general environment of inference rules (rather than just resolution, a variation of critical-pair completion) than available in our model.

The set-up of Rydeheard and Stell (1987) was adapted by Stell (1992, 1994), where a generalization of 2-categories, sesqui-categories, is used for modeling term rewriting. This enables one to distinguish reductions of different length. For left-linear TRS one then can characterize the critical pairs categorically as so-called critical spans; but not for general TRS (there are critical spans that are not critical pairs).

By Jay (1991), already completed (hence confluent) reduction systems are modeled by so-called confluent categories, with confluent orders as objects and order-preserving functions as morphisms. He obtains a categorical semantics for the reduction and a functorial relationship between the operational semantics (the reduction process computing normal forms) and the denotational semantics (obtained by identifying any term with its reducts, in particular its normal forms). The model is formulated without 2-categories, to avoid the conceptual overhead of equations on the 2-cells and having to establish coherence theorems. This also motivated us to model the critical-pair/completion of reduction systems by “plain” categories (completion is not treated in Jay’s approach, as he only considers confluent systems).

There are two more recent applications of categorical methods and constructs in the area of reduction systems.

First, Lüth and Ghani (1996) used monads for modeling TRS and thereby obtained a purely categorical proof of the modularity of confluence for the disjoint union of TRS.

Second, Melliès (1996) distinguished between external and internal factorizations to separate the “efficient” part of a computation in an axiomatic rewriting system from its “junk”. A category of external derivations enjoying several categorically elegant properties is thereby obtained.

4. Categorical Constructs

For details on basic category theory, we refer to, among others, Mac Lane (1971), Goldblatt (1984) and Lambek and Scott (1986) and Adámek et al. (1990). The only slightly non-standard notion used is that of a polynomial category, on which more details can be found from Lambek and Scott (1986).

The only original item in this section is the presentation of what we call reversed limits (resp. reversed colimits), which are limits (resp. colimits) in special slice categories. We prove one essential result pertaining to them here; this result enables us to formulate the critical pair lemma in a categorical way. More on reversed limits and colimits can be found by Stokkermans (1995a, b).
4.1. Basic Category Theory

The definitions of the notions of category, slice or comma category, functor, diagram, cone, limit and colimit can be found in the standard texts mentioned above; here we only define polynomial categories. One remark on notation: we will generally use capital letters $A, B, \ldots$ for the objects of a category, single arrows $\to$ for the arrows within a category, and double arrows $\Rightarrow$ to indicate functors.

**Definition 4.1. (Freely Generated Category)** The freely generated category $F(G)$ by the (directed) graph $G$ is the smallest category with objects as the nodes of $G$ and arrows as all edges from $G$ and closed under associativity of arrows and the presence of all identity arrows.

**Definition 4.2. (Polynomial Category)** For any category $C$ and any arrow $f: A \to B$ between two objects $A$ and $B$ in $C$ such that $f$ is not yet among the arrows of $C$, the polynomial category $C[f]$ is denoted as the category freely generated from the graph $G'$, where $G'$ is the graph formed by adding the arrow $f$ to the underlying graph $G$ of $C$.

We will use these polynomial categories for modeling completion: the addition of a rewrite rule or polynomial (a pattern in the sense of Buchberger, 1987) corresponds to the construction of a polynomial category.

4.2. Reversed Limits

Two new notions are introduced. They are natural extensions of the notions of limit and colimit, respectively, reversing the morphism establishing the universal property. Limit and reversed limit form boundaries between which the cones of a diagram range. Analogously, colimit and reversed colimit form boundaries for the cocones.

The reason for introducing these notions is that the existence of normal forms in the context of reduction systems can be expressed in a very natural way with the help of special reversed colimits, so-called normalizers. Moreover, the equivalence of strong confluence and the Church-Rosser property can be formulated as a statement on reversed colimits (cf. Theorem 4.1).

The following definition expresses that a reversed limit of a given diagram $\Delta$ factors uniquely through all cones for $\Delta ; I$ is an index set for the objects in a given diagram and $J$ an index set for the arrows between them.

**Definition 4.3. (Reversed Limit)** A reversed limit for a diagram $\Delta = \{D_i, g_j \mid i \in I, j \in J\}$ is a $\Delta$-cone $\{f_i : C \to D_i \}_{i \in I}$ with the property that for any other $\Delta$-cone $\{f'_i : C' \to D_i \}_{i \in I}$ there is exactly one morphism $f : C \to C'$ such that the diagram

\[
\begin{array}{ccc}
C & \xrightarrow{f} & C' \\
\downarrow f' & & \downarrow f' \\
D_i & \xrightarrow{g_j} & D_i \\
\end{array}
\]

commutes for every object $D_i$ in $\Delta$. 
Conversely, all cocones for a diagram \( \Delta \) factor uniquely through its reversed colimit.

**Definition 4.4. (Reversed Colimit)** A reversed colimit for a diagram \( \Delta = \{D_i, g_{ij} \mid i \in I, j \in J \} \) is a \( \Delta \)-cocone \( \{f_i: D_i \rightarrow C \}_{i \in I} \) with the property that for any other \( \Delta \)-cocone \( \{f'_i: D_i \rightarrow C' \}_{i \in I} \) there is exactly one morphism \( f: C' \rightarrow C \) such that the diagram

\[
\begin{array}{ccc}
D_i & \xrightarrow{f_i} & C' \\
\downarrow{f'_i} & & \downarrow{f} \\
C & & C
\end{array}
\]

commutes for every object \( D_i \) in \( \Delta \).

Both reversed limits and colimits are determined uniquely up to isomorphism.

Initial and terminal objects are the most trivial examples of reversed limits and colimits, respectively. An initial object is a reversed limit of the empty diagram, while a terminal object is a reversed colimit of the empty diagram.

**Definition 4.5. (Normalizer)** A normalizer (relative to a given object \( A \)) is a reversed colimit of the diagram consisting of the object \( A \).

Note that the corresponding limit (or colimit) would be the object \( A \) itself, with its identity morphism as limiting (or colimiting) arrow.

A normalizer of an object \( A \) in a category \( C \) is a terminal object in the category \( A \downarrow C \). The term normalizer is chosen here because in the category \( \kappa \text{CPC} \) which we will define in Section 5, the normalizer of an object \( A \) corresponds to its normal form.

Note that if a category \( C \) contains a terminal object, then all reversed colimits (for arbitrary diagrams) will correspond to that terminal object, as will follow from Theorem 4.1. Therefore, the consideration of such reversed colimits is only interesting in categories that do not have a terminal object—or, in the context of reduction, in those cases where not all objects have one and the same normal form (the relevant representation of the categorical constructs will be presented in Section 5).

The notion dual to normalizer corresponds to an object which cannot be made more complex (with respect to the reduction relation defining the arrows in the category \( \kappa \text{CPC} \)). It is as far away from the normalizer of an object as possible and will therefore be called denormalizer. (Of course, it would be the normalizer of the given object in the dual category \( C^{op} \) which is obtained from \( C \) by reversing all arrows.)

**Definition 4.6. (Denormalizer)** A denormalizer (relative to a given object \( A \)) is a reversed limit of the diagram consisting of the object \( A \).

Again, there is an alternative formulation: a denormalizer of an object \( A \) in a category \( C \) is an initial object in the category \( C \downarrow A \).

**Definition 4.7. (Reversed Product and Coproduct)** A reversed product of objects \( A \) and \( B \) is a reversed limit of the discrete diagram consisting of \( A \) and \( B \). A reversed coproduct of \( A \) and \( B \) is a reversed colimit of that diagram.
One could construct reversed limits for other diagrams, but the following result shows that normalizers and denormalizers are the basic types of reversed colimits and limits of connected diagrams, respectively.

**Theorem 4.1.** For any category \( C \) and any connected non-empty diagram scheme \( \Delta \), the existence of all normalizers is equivalent to the existence of all reversed colimits for the diagram scheme \( \Delta \).

**Proof.** Directly from the following two lemmas.

I should remark here that more or less the same result (using slightly differing terminology) was independently communicated to me by Barry Jay; the formulation and the proof appearing here are mine.

The fact that the existence of all normalizers is equivalent with the existence of all reversed pushouts (a direct corollary of Theorem 4.1) corresponds to the equivalence between confluence and the Church–Rosser property for reduction systems.

**Lemma 4.1.** The existence of all normalizers guarantees the existence of all reversed colimits of non-empty diagram schemes that are connected (directed) graphs.

**Proof.** In order to prove the existence of all reversed colimits of non-empty diagram schemes that are connected (directed) graphs, it suffices to show that all reversed colimits of arbitrary instantiations of such diagram schemes exist. Consider, therefore, a non-empty diagram \( \Delta \) of which all objects are connected by morphisms (i.e. \( \Delta \) is a connected directed graph). For any two objects \( A, B \) such that we have a morphism \( f: A \to B \) in \( \Delta \), we construct:

(i) the normalizer \( A' \) together with a morphism \( a: A \to A' \) and, by definition, the unique arrow \( g: B \to A' \) such that \( g \circ f = a \);

(ii) the normalizer \( B' \) together with a morphism \( b: B \to B' \) and, because of \( g \), the unique arrow \( h: A' \to B' \) such that \( h \circ g = b \);

(iii) the unique arrow \( k: B' \to A' \) obtained from the morphism \( b \circ f: A \to B' \), such that \( k \circ b \circ f = a \).

Now, clearly \( A' \) and \( B' \) are isomorphic by \( h \) and \( k \). That \( k \circ h = \text{id}_{A'} \) and \( h \circ k = \text{id}_{B'} \) can be seen as follows: \( k \circ h \circ a = k \circ h \circ g \circ f = k \circ b \circ f = a \); as \( A' \) is its own normalizer, \( \text{id}_{A'} \) is the only morphism \( f' \) such that \( f' \circ a = a \), but we have just shown that \( k \circ h \) fulfills the same property. Likewise, \( h \circ k = \text{id}_{B'} \) follows from the equation \( h \circ k \circ b \circ f = h \circ a = h \circ g \circ f = b \circ f \) and the fact that \( B' \) is its own normalizer.

By continuing this process, we find that the normalizers of the objects in \( \Delta \) are all isomorphic, say to a reversed colimit \( C \). By the commutativity of all relevant triangles it now follows that \( C \) is the reversed colimit of \( \Delta \). □

We will use the above result, stating in particular that the existence of all normalizers guarantees the existence of all reversed pushouts, in the following section on the categorical formulation of CPC algorithms.

The above lemma, together with the following, immediate one, verifies the truth of Theorem 4.1.
Lemma 4.2. If the existence of all colimits for an arbitrary non-empty diagram scheme \( \Delta \) is guaranteed in a given category \( C \), then all normalizers exist in \( C \).

Proof. To construct the normalizer of an arbitrary object \( A \) in \( C \), simply construct the reversed colimit of the diagram \( \Delta \) with all objects equal to \( A \) and all morphisms \( 1_A \). \( \square \)

This means that normalizers can be considered as special instances of other reversed colimits, in particular reversed pushouts, reversed coproducts, and reversed coequalizers. (Also, we can consider denormalizers as special instances of reversed pullbacks, reversed products, and reversed equalizers, respectively.)

5. The Categorical Framework for CPC Algorithms

In this section we detail the requirements on the constituents of the categorical model for CPC algorithms. A remark on notation: we will always denote objects of a category by uppercase, arrows by lowercase, and functors by Greek letters.

I should stress here that the categorical model was inspired by the ideas on an axiomatization based on patterns, multipliers, and replacements brought forward by Buchberger [1987]. We show here that with a slight modification in the concept of replacement (namely by viewing this as an operation embedding terms at given contexts), this basic operation can be formulated in a functorial way.

The categorical constructions we present below are closely interrelated by means of functors which have a natural interpretation in the domain of the individual algorithms.

All constructions are necessary in order to express the basic ideas underlying CPC algorithms as generally as possible, while not including superfluous machinery.

The following picture can serve as a guideline for the basic interrelationships between the categories which will be constructed, namely \( \mathcal{CPC} \), \( \mu \mathcal{CPC} \), \( \eta \mathcal{CPC} \), \( \kappa \mathcal{CPC} \), and \( \kappa \mu \mathcal{CPC} \).

5.1. THE OBJECTS

The objects on which critical-pair completion is performed will be modeled by (labeled) trees. It will be shown in the next section that for all the three main critical-pair completion procedures, the objects under consideration can be viewed as such.

The main property of these objects is that any node of a labeled tree representing an object represents a subobject, and that some of the leaf nodes are "variable objects". In order to identify subobjects within an object, we introduce operators \( \pi \) indexed with places (as strings of positive numbers, cf. the Dewey notation for terms); these will be essential in incorporating the embedding operators later on.

Definition 5.1. (Object) An object is a tree \( A \) whose nodes are labeled by strings of positive integers \( \langle k_1 k_2 \ldots k_n \rangle \). The labels of all nodes of a tree (including the root node, labeled \( () \), and the leaf nodes) form a set of places \( P(A) \) satisfying the following conditions (where all \( k_i \) are positive natural numbers):

\[
\begin{align*}
\text{(π-1)} & \text{ if } \langle k_1 k_2 \ldots k_{n-1} k_n \rangle \in P(A), \text{ for } n \geq 1, \text{ then } \langle k_1 k_2 \ldots k_{n-1} \rangle \in P(A); \\
\text{(π-2)} & \text{ if } \langle k_1 k_2 \ldots k_{n-1} k_n \rangle \in P(A), \text{ for } n \geq 1, \text{ then } \langle k_1 k_2 \ldots k_{n-1} k \rangle \in P(A), \text{ for any } 1 \leq k < k_n.
\end{align*}
\]
For any place \( u \in P(A) \), the subobject of \( A \) occurring at \( u \) will be denoted \( A_u \).

For any two places \( u \) and \( v \), \( u \leq v \) will denote the situation that \( u \) is an initial substring of \( v \). We will use \( u \perp v \) for the situation that \( u \) and \( v \) are disjoint places (so \( u \perp v \) iff \( u \not\preceq v \) and \( v \not\preceq u \)).

For an arbitrary object \( A \), \( V(A) \) denotes its set of variable subobjects; any variable object is an object \( V \) with \( P(V) = \{h\} \).

Finally, we introduce, for each finite string of positive natural numbers \( u = (k_1k_2 \ldots k_n) \), operators \( \pi_u \), defined on those objects \( A \) such that \( u \in P(A) \), which map \( A \) to \( A_u \). It is clear that w.r.t. a given object \( A \), these operators satisfy \( \pi_v(A) = \pi_u \circ \pi_{v_2}(A) \) if \( v \) is the concatenation of \( u_1 \) followed by \( u_2 \). (Indeed, one can build a category of places, \( \kappa \text{CPC} \), by viewing these operators as arrows, the view taken by Stokkermans (1995a).)

The universe of objects under consideration in the general framework will be denoted by \( V \); its actual nature of course depends on the specific context of a given CPC procedure.

In the general context of CPC, we will have to deal with objects modulo a certain equivalence relations \( \mathcal{F} \) (e.g. in the case of the completion of term rewriting systems modulo associativity and commutativity, or in general in the case of the construction of Gr"{o}bner bases where we work modulo equalities on polynomials in non-standard form).
Definition 5.2. (Universes) For the universe of (canonical representatives of) the equivalence classes of $V$ under $F$ we will use $U$. All variable objects are (as canonical representatives of their equivalence class) elements of $U$.

5.2. Operations on Objects

We will introduce two fundamental types of operations on the objects introduced above, namely multiplying and embedding. Later, we will see that these operations are functorial in nature; for now, we will concentrate on their interaction with the place operators.

Multiplier operators, multipliers for short, are based on the variable (sub)object. They essentially replace a variable object by another object, at all places where the given variable object appears. The general definition provides for such replacings of finitely many variable objects.

Definition 5.3. (Multiplier) A multiplier $\mu$ is defined on a finite set of variable objects, denoted $V_\mu$. Any $V \in V_\mu$ is mapped to an object $\mu(V_\mu) \in U$. For any $V \notin V_\mu$, $\mu(V) = V$.

It is straightforward to extend any multiplier $\mu$ to a mapping on all objects by requiring that it replaces all relevant occurrences of variable objects; we require the following two properties for all multipliers, for any two objects $A, B$:

\begin{itemize}
  \item \((\mu-1)\) if $A = \mu(B)$ then $P(A) = P(B) \cup \{ (k_1 \ldots k_n l_1 \ldots l_m) \mid B(k_1 \ldots k_n) = V \in V_\mu \text{ and } \langle l_1 \ldots l_m \rangle \in P(\mu(V)) \}$;
  \item \((\mu-2)\) for any place $u \in P(A)$ we have $\pi_u(\mu(A)) = \mu(\pi_u(A))$.
\end{itemize}

Multipliers can be composed, and indeed they generate a category, called $\mu\text{CPC}$, as follows.

Definition 5.4. (Composition of Multipliers) Let $\mu_1, \mu_2$ be multipliers, then their composition $\mu_1 \circ \mu_2$ is defined on variable objects $V$ by:

\begin{itemize}
  \item if $V \in V_{\mu_2}$ then $\mu_1 \circ \mu_2(V)$ is obtained from the extension of $\mu_1$ to the object $\mu_2(V)$;
  \item if $V \notin V_{\mu_2}$ then $\mu_1 \circ \mu_2(V) = \mu_1(V)$.
\end{itemize}

For arbitrary objects, $\mu_1 \circ \mu_2$ is again the straightforward extension of the above.

Definition 5.5. (\(\mu\text{CPC}\)) $\mu\text{CPC}$ is the category with as objects all elements of $U$; there is an arrow $\mu : A \rightarrow B$ iff for a multiplier $\mu$ we have: $\mu(A) = B$.

That $\mu\text{CPC}$ indeed forms a category can be seen as follows:

\begin{itemize}
  \item the identity arrows are given by the empty multiplier $(V_\mu = \emptyset)$;
  \item for two arrows $\mu_2 : A \rightarrow B$ and $\mu_1 : B \rightarrow C$, the arrow $\mu_1 \circ \mu_2$ transforms $A$ into $C$;
  \item the associativity and identity rules are straightforward.
\end{itemize}

Embedder operators, embedders for short, are based on plugging in objects at given places of larger objects (thereby deleting whatever subobject occurred there before).
Definition 5.6. (Embedder) An embedder $\eta_{A,u}$ maps an object $B$ to the object $\eta_{A,u}(B)$ obtained from $A$ by replacing its subobject $A_u$ by $B$.

We require the following two properties for all embedders $\eta_{A,u}$:

($\eta$-1) for any objects $A, B$ and place operator $\pi_u$: $\pi_u(\eta_{A,u}(B)) = B$;
($\eta$-2) for any object $A$ and place operator $\pi_u$: $\eta_{A,u}(\pi_u(A)) = A$.

The first of these rules says that if one plugs in $B$ at a given place $u$ of $A$, the resulting object $\eta_{A,u}(B)$ will have the subobject $B$ at position $u$. The second says that embedding the subobject occurring at position $u$ of $A$ into $A$ at $u$ will result in $A$ again. Together, $\eta$-1 and $\eta$-2 describe the result of an embedding completely: let $\eta_{A,u}$ be an arbitrary embedder, and let $C$ be defined as $\eta_{A,u}(B)$, for an arbitrary object $B$. From $\eta$-1 we see $\pi_u(C) = B$. This also fixes all subobjects occurring at places $v$ such that $u$ is an initial subsequence of $v$. For all other places $w$, $\eta$-2 implies $\pi_w(C) = \pi_w(A)$.

Moreover, $\eta$-1 and $\eta$-2 imply that for any two embedders $\eta_1 = \eta_{A,u}$ and $\eta_2 = \eta_{B,v}$ with the property that for some object $C$, $\eta_1(C) = \eta_2(C)$, we either have that both $\eta_1$ and $\eta_2$ are equal on arbitrary objects, or both $A \neq B$ and $u \perp v$.

Embedders can be composed, as follows.

Definition 5.7. (Composition of Embedders) Let $\eta_{A,u}, \eta_{B,v}$ be embedders, then their composition $\eta_{A,u} \circ \eta_{B,v}$ is defined on arbitrary objects $C$ as $\eta_{A,u}(\eta_{B,v}(C))$.

Again, the embedders generate a category $\eta_{\mathcal{CPC}}$.

Definition 5.8. ($\eta_{\mathcal{CPC}}$) $\eta_{\mathcal{CPC}}$ is the category with as objects all elements of $\mathcal{U}$; for every $C \in \mathcal{U}$ and $u \in P(C)$ there is an arrow $\eta_{C,u}: A \rightarrow B$ iff for the embedder $\eta_{C,u}$ we have: $\eta_{C,u}(A) = B$.

That $\eta_{\mathcal{CPC}}$ indeed is a category can be seen as follows:

- the identity arrow on a given object $A$ is given by the embedder $\eta_{A,()}$ (embedding $A$ at its root position);
- for two arrows $\eta_{D,u}: A \rightarrow B$ and $\eta_{E,v}: B \rightarrow C$, we have $(\eta_{E,v} \circ \eta_{D,u})(A) = \eta_{E,v}(\eta_{D,u}(A)) = \eta_{E,v}(B) = C$;
- the identity and associativity rules are straightforward.

5.3. Patterns

We are now ready to incorporate the basic notion of reduction relations in our framework, namely that of the basic patterns, which describe how objects can be reduced to other ones. For that purpose, we now build the reduction category $\mathcal{CPC}$.

In the general context, we are given a universe $\mathcal{U}$ of objects as defined in Section 5.1, equipped with a partial order $>_{\mathcal{X}}$, and certain elementary reduction rules (derived from equalities between polynomials, equivalences between terms etc., depending on the context) which respect this order $>_{\mathcal{X}}$. 
Definition 5.9. (pattern) A pattern $p$ is a pair of objects $(A, B)$ such that $A >_\kappa B$.

In the context of CPC, we will always have a finite set of patterns given at the start of the procedure. This finite set will give rise to our basic reduction category $\mathcal{CPC}$ and its canonized version $\kappa\mathcal{CPC}$ by generating additional reductions between objects, based on the multipliers and the embedders.

We add one more demand on the behavior of the multipliers w.r.t. patterns here:

($\mu$-3) for any pattern $p: A \rightarrow B$, we have: $\forall \mu_1, \mu_2, \text{if } \mu_1(A) = \mu_2(A) \text{ then } \mu_1(B) = \mu_2(B)$.

To guarantee $\mu$-3, it suffices to demand that the right-hand side of a pattern $p$ does not contain any variables not occurring at the left: $V(B) \subseteq V(A)$. Property $\mu$-3 ensures that two multipliers having the same effect on the left-hand side of a pattern also have the same effect on the right-hand side, which is essential in proving the critical pair lemma.

5.4. THE FUNDAMENTAL CRITICAL-PAIR COMPLETION CATEGORY

We will introduce two categories in order to encapture the reduction process. In the first, called $\kappa\mathcal{CPC}$, the patterns are used to generate the arrows by means of the multiplier and embedder operators provided in the given context; these arrows respect the partial order $>_\kappa$ and model the possible reductions.

It can then be required (which is necessary in the context of CPC) that multipliers are endofunctors on $\kappa\mathcal{CPC}$.

However, in order to model the Gröbner basis case or rewriting modulo equivalences, we have to take the fact into account that embedding a small object in a larger one can alter the structure of the latter (for instance if we plug in a monomial with a power product that is already represented in the bigger context).

In order to capture this, we have to add objects of a different, more general nature, which are related to those of $\kappa\mathcal{CPC}$ by means of a given equivalence relation $\mathcal{F}$, and which can be transformed back to objects of $\kappa\mathcal{CPC}$ by a canonization operator, which takes the result of an embedder operation and “takes it back” into $\kappa\mathcal{CPC}$.

These more general objects will be part of a larger category $\mathcal{CPC}$, which will be related to $\kappa\mathcal{CPC}$ by the embedder operators (which will serve as functors from $\kappa\mathcal{CPC}$ into $\mathcal{CPC}$) and a canonizer functor $\kappa$.

The category $\mathcal{CPC}$ is the category freely generated from the arrows obtained by applying multipliers and embedders to the original patterns, but such that all arrows with the same source and target are identified.

Definition 5.10. ($\kappa\mathcal{CPC}$) The reduction graph $\kappa\mathcal{CPC}$ has as nodes the elements of $\mathcal{U}$. There is an edge from an object $A$ to an object $B$ iff there is a pattern $p = (L, R)$, a multiplier $\mu$ and an embedder $\eta$ such that $A = \eta(\mu(L))$ and $B = \eta(\mu(R))$.

The reduction category $\mathcal{F}(\kappa\mathcal{CPC})$ (denoted $\kappa\mathcal{CPC}$ in the remainder of the paper) is the freely generated category with the reduction graph $\kappa\mathcal{CPC}$ as underlying graph, but such that all arrows with equal source and target are identified.
The reduction category \( \kappa \text{CPC} \) models a Noetherian reduction relation. The objects correspond to the elements that are to be reduced; the arrows indicate (possible) reductions. In order that this functions properly, we use the partial order \( >_\kappa \) on the objects of \( \kappa \text{CPC} \), and require that the arrows respect this order; then no “reduction loops” can occur:

\[(\kappa \text{-1}) \quad \kappa \text{CPC} \text{ is a preorder;}
\]

\[(\kappa \text{-2}) \quad \text{for every } f: A \to B \text{ in } \kappa \text{CPC, } A \geq_\kappa B.
\]

We also require that the multipliers are endofunctors on \( \kappa \text{CPC} \):

\[(\mu \text{-4}) \quad \text{all multipliers are faithful covariant endofunctors on } \kappa \text{CPC}.
\]

As mentioned above, in general, the embedders are not endofunctors on \( \kappa \text{CPC} \), and we now introduce a “larger” category, called CPC, which contains several representations of the same object in \( \kappa \text{CPC} \). In other words, the objects of \( \kappa \text{CPC} \) are from now on to be regarded as (the canonical representatives of) equivalence classes w.r.t. a given set of equations \( \mathcal{F} \). We require:

\[(\kappa \text{-3}) \quad \text{the reduction relation defining } \kappa \text{CPC is } \mathcal{F}\text{-compatible (in the sense of Definition 9.1 by Peterson and Stickel, 1981).}
\]

Note that \( >_\kappa \) is not required to respect \( \mathcal{F} \). (Indeed, it will turn out that in the case of Gröbner bases, it is impossible to guarantee this.)

The category CPC has as objects the elements of a universe \( \mathcal{V} \) we work in; the objects of \( \kappa \text{CPC} \) are the elements of \( \mathcal{U} = \mathcal{V}/=_x \). We will denote objects of CPC by \( A \) and those of \( \kappa \text{CPC} \) by \([A]\) (as they are equivalence classes of objects in CPC). The objects in CPC in canonical form are also denoted as \([A]\); these are the only objects shared by \( \kappa \text{CPC} \) and CPC. The place operators \( \pi_u \) are defined on CPC as on \( \kappa \text{CPC} \): for any \( A \) in CPC for which the place \( u \) is defined, \( \ldots u(A) = A_u \).

Now to the arrows of CPC. Any \( f: [A] \to [B] \) in \( \kappa \text{CPC} \) induces an \( f: [A] \to [B] \) in CPC. Arrows are preserved under the embedder functors \( \eta: \kappa \text{CPC} \Rightarrow \text{CPC} \): for any \( f: [A] \to [B] \) in CPC and any such \( \eta \), there is a \( \eta(f): \eta([A]) \to \eta([B]) \) in CPC. Finally, for any objects \( A, B \) with \( A =_x B \), there are inverse arrows \( e_{AB}: A \to B \) and \( e_{BA}: B \to A \). Summarizing:

\[(\kappa \text{-4}) \quad \text{CPC is a category;}
\]

\[(\kappa \text{-5}) \quad \text{if } A =_x B, \text{ then there are arrows } e_{AB}: A \to B \text{ and } e_{BA}: B \to A \text{ in CPC such that } e_{BA} \circ e_{AB} = 1_A \text{ and } e_{AB} \circ e_{BA} = 1_B.
\]

As stated, we require that the embedders are functors from \( \kappa \text{CPC} \) to CPC. The precise nature of the embedders depends on the specific CPC algorithm considered. The general requirement is:

\[(\eta \text{-3}) \quad \text{every embedder } \eta \text{ is a functor from } \kappa \text{CPC into CPC}.
\]

The category \( \eta \text{CPC} \) has as objects all elements of \( \mathcal{V} \) (like CPC), and as arrows the embedders: for every \( \eta \) and \( A \) there is an arrow from \( A \) to \( \eta(A) \). As explained before,
embedders are given by an object $[B]$ of $\kappa\text{CPC}$ and a place $i$ within $[B]$: intuitively, $\eta_{[B],i}$ plugs in a given object $A$ at $i$ within $[B]$, resulting in the object $\eta_{[B],i}(A)$ of CPC.

The canonizer functor $\kappa$, an endofunctor on CPC, maps all objects to their canonical form (which also is an object in $\kappa\text{CPC}$) and every arrow to an arrow between such canonical forms. The arrows in the image of CPC under $\kappa$ need not respect $>_{\kappa}$; that is why we cannot define $\kappa$ as a functor from CPC into $\kappa\text{CPC}$ and why the introduction of CPC is necessary. We require of $\kappa$:

(\kappa-6) $\kappa$ defines an endofunctor on CPC;
(\kappa-7) $\kappa(\mathcal{V}) \subseteq \mathcal{U}$.

5.5. two more auxiliary categories

We need two more categories to formulate the general CPC algorithm. The first, $\kappa\eta\text{CPC}$, will enable us to bring patterns in the “right” form, while the second, $\kappa\mu\text{CPC}$, is used to determine critical pairs.

First, the category of increasing embedders, $\kappa\eta\text{CPC}$. Its objects are all arrows of $\kappa\text{CPC}$ (the reductions) and its arrows the so-called increasing embedders: for $f: [A_1] \to [A_2]$ and $g: [B_1] \to [B_2]$ in $\kappa\text{CPC}$ we have an arrow in $\kappa\eta\text{CPC}$ from $f$ to $g$ iff there is an embedder $\eta$ such that: $\kappa\eta([A_1]) = [B_1], \kappa\eta([A_2]) = [B_2], [A_1] >_{\kappa} [A_2], [B_1] >_{\kappa} [B_2]$ (the inequalities follow from $\kappa-2$) and $[A_1] >_{\kappa} [B_1]$. This category brings patterns in the right form: such that the left-hand side is as small as possible but still greater than the right-hand side. We require:

($\kappa\eta-1$) $\kappa\eta\text{CPC}$ is a category;
($\kappa\eta-2$) $\kappa\eta\text{CPC}$ has normalizers.

Second, the category of equivalence preservers, $\kappa\mu\text{CPC}$. Its objects are those of $\kappa\text{CPC}$ and its arrows all operations preserving the equivalence underlying the reduction relation: the multipliers, embedders and canonizer. (These operations preserve equivalence in the sense that if $[A] \leftrightarrow [B]$ then also $\mu([A]) \leftrightarrow \mu([B])$ and $\eta([A]) \leftrightarrow \eta([B])$ for any $\mu$ and $\eta$.) In $\kappa\mu\text{CPC}$ we find the critical pairs as the weak coproducts of the left-hand sides of two patterns. We have:

($\kappa\mu$) $\kappa\mu\text{CPC}$ is a category.

The arrows of $\kappa\mu\text{CPC}$ do not necessarily respect the ordering $>_{\kappa}$; they are arrows between canonized objects but need not be canonized themselves (unlike arrows in $\kappa\eta\text{CPC}$). Also, $\kappa\mu\text{CPC}$ is not required to have any coproducts. If it has no coproducts, no critical pairs exist w.r.t. the reduction relation.

We can now characterize critical pairs categorically, thereby obtaining a uniform formulation for the concept for all CPC algorithms.

**Definition 5.11. (Critical Normalizers)** Given a finite set $P$ of patterns $A_i \to B_i$, with $1 \leq i \leq n$ and the corresponding category $\kappa\text{CPC}$ such that every $p_i: A_i \to B_i$ is its own normalizer in $\kappa\text{CPC}$. Then, a reversed pushout diagram in $\kappa\text{CPC}$.
is called a critical normalizer for the reduction system defined by $\kappa \text{CPC}$ iff $C$ is a weak coproduct of $A_j$ and $A_k$ ($1 \leq j, k \leq n$) in $\kappa \eta \mu \text{CPC}$.

5.6. Generalizing the Critical-pair Lemma

We now concentrate on bringing the critical-pair lemma in a categorical form. The critical-pair lemma states that local confluence is equivalent to critical-pair confluence. As for the other two main confluence lemmas pertaining to reduction systems, we have already seen that the equivalence between the Church–Rosser property and confluence is a consequence of Theorem 4.1, while the equivalence between confluence and local confluence (Newman’s lemma) only requires the Noetherianity of the reduction relation.

**Lemma 5.1.** If the reduction system defining $\kappa \text{CPC}$ is Noetherian, the unique normal form of an object $A$ (if it exists) is the normalizer $\Delta$ of $A$ in $\kappa \text{CPC}$.

**Proof.** If $\Delta$ is the unique normal form of $A$, then for any $B$ such that $A$ reduces to $B$, $B$ can be reduced to $\Delta$. This shows that $\Delta$ is the normalizer of $A$ in $\kappa \text{CPC}$. Conversely, if $\Delta$ is the normalizer of $A$ in $\kappa \text{CPC}$, any object $B$ such that $A$ reduces to $B$ has a reduction to $\Delta$. As the reduction relation $\rightarrow$ is Noetherian, $\Delta$ then is the unique normal form of $A$. \(\Box\)

Now, let also a category of multipliers $\mu \text{CPC}$, a category of embedders $\eta \text{CPC}$, and a canonizer functor $\kappa : \text{CPC} \Rightarrow \text{CPC}$ be defined subject to the conditions of Section 5.2–5.4. Let $\eta \mu \text{CPC}$ and $\kappa \eta \mu \text{CPC}$ be defined as in Section 5.5.

Recall that critical normalizers are weak coproducts in $\kappa \eta \mu \text{CPC}$ of left-hand sides of patterns defining $\kappa \text{CPC}$ (possibly added during the completion process).

We say that the completion of the category $\kappa \text{CPC}$ is terminated successfully (by the categorical CPC algorithm presented in the next subsection) if all objects in $\kappa \text{CPC}$ have a unique normal form—i.e. a normalizer, because of the previous lemma. By Theorem 4.1, this is equivalent to the statement that all reversed pushout diagrams in $\kappa \text{CPC}$ have a reversed colimit. Given that no reductions have infinite length (because of the requirement that the reduction system be Noetherian), this gives us the following categorical counterpart to the critical-pair lemma.

**Theorem 5.1. (Categorical Critical-pair Lemma)** The completion of $\kappa \text{CPC}$ is terminated successfully if all critical normalizers have a reversed pushout in $\kappa \text{CPC}$.

**Proof.** Recall that the critical-pair lemma asserts the equivalence of local confluence with critical-pair confluence. So we only need to consider possible violations of local confluence.

Let $A$ be any object in $\kappa \text{CPC}$ and $f = (p_1, \mu_1, \eta_1)$ and $g = (p_2, \mu_2, \eta_2)$ two arbitrary arrows with domain $A$, both by one reduction step.
We must show that the diagram

\[ \begin{array}{c}
A \\
\kappa \mu_1(R_1) & \kappa \mu_2(R_2)
\end{array} \]

\[ \xymatrix{ & B \\
\kappa \mu_4(R_1) & \kappa \mu_5(R_2) \ar[ll]^{h} \ar[rr]_{k} & & B }
\]

can be closed. So, we have to find arrows with domain \( \kappa \mu_1(R_1) \) and \( \kappa \mu_2(R_2) \), respectively, and with equal codomain. Consider the object \( B \) in \( \kappa \mu \text{CPC} \) defined as a weak coproduct of \( L_1 \) and \( L_2 \) such that we have a \( \langle \eta_3, \mu_3 \rangle: B \rightarrow A \) in \( \kappa \mu \text{CPC} \). As \( B = \eta \mu_4(L_1) = \eta \mu_5(L_2) \), by the assumption we have the following “reversed pushout” diagram in \( \kappa \text{CPC} \):

As \( l \) is the arrow modeling the reduction \( \kappa \mu_4(R_1) \rightarrow B, \mu_3(l) \) is the arrow modeling the reduction \( \mu_3 \kappa \mu_4(R_1) \rightarrow \mu_3(B) \) (and as multipliers are endofunctors on \( \kappa \text{CPC} \), the left-hand side may be written as \( \kappa \mu_4 \eta_4(R_1) \)).

Furthermore, \( \eta_3 \mu_3(l) \) is an arrow in CPC from \( \eta_3 \kappa \mu_4 \eta_4(R_1) \) to \( \eta_3 \mu_3(B) \). Since in CPC we have arrows between objects in the same equivalence class (which we can guarantee thanks to the compatibility of the equivalence and the reduction relations, \( \kappa \)-3), in all we obtain an arrow from \( \kappa \mu_4 \eta_4(R_1) \) to \( \kappa \mu_4 \eta_5(B) \).

But the left-hand side is equivalent to \( \kappa \mu_1(R_1) \), and as the same argument applies to \( m: \kappa \mu_5(R_2) \rightarrow B \), we also obtain an arrow \( \eta_3 \mu_3(m) \) from \( \kappa \mu_2(R_2) \) to \( \kappa \mu_5 \eta_3(B) \).

Composing \( \eta_3 \mu_3(l) \) and \( \eta_5 \mu_3(m) \) with \( f \) and \( g \), respectively, concludes the proof. \( \square \)

5.7. THE CATEGORICAL CPC ALGORITHM

After the above preparations, we are now ready to formulate the CPC algorithm in the language of category theory, thereby obtaining a uniform framework for the three main CPC algorithms.

Input:

- a set of patterns \( P = \{ (l_1, r_1), \ldots, (l_n, r_n) \} \), that is, pairs \( (l_i, r_i) \) of elements of a universe \( U \);
- a partial ordering \( >_\kappa \) on \( U \), such that \( l_i >_\kappa r_i \) for all patterns in \( P \);
- a reduction category \( \kappa \text{CPC} \) (generated by \( P \) and \( >_\kappa \)), fulfilling the conditions \( \kappa \)-1, \( \kappa \)-2, \( \kappa \)-3, \( \kappa \)-4, \( \kappa \)-5, \( \kappa \)-6 and \( \kappa \)-7;
- classes of functors \( \mu, \kappa, \) and \( \eta \), satisfying \( \mu \)-1, \( \mu \)-2, \( \mu \)-3, \( \mu \)-4, \( \eta \)-1, \( \eta \)-2, \( \eta \)-3, \( \kappa \)-3, \( \kappa \)-4, \( \kappa \)-5, \( \kappa \)-6 and \( \kappa \)-7.
Output:

a reduction category \( \kappa \text{CPC} [f_1, \ldots, f_m] \) containing the original reduction category \( \kappa \text{CPC} \) as a subcategory and generated from it by the arrows \( \{f_1, \ldots, f_m\} \), such that all objects in the category have a normalizer. The \( f_i \) correspond to the additional “rewrite rules” or “polynomials” added during the completion, and together with the original patterns they constitute the completed system.

Algorithm:

\[
P := \{(l, r) \mid l \rightarrow r \text{ is the normalizer in } \kappa \eta \text{CPC} \text{ of some } l_i \rightarrow r_i, 1 \leq i \leq n\};
\]
\[
CN := \{(l_i, l_j) \mid (l_i, r_i), (l_j, r_j) \in P\};
\]

while \( CN \neq \emptyset \) do (\( (l_i, l_j) \): = a pair in \( CN \); \( CN := CN - \{(l_i, l_j)\} \);

\( (\phi_1, \phi_2) := \) coproduct of \( l_i \) and \( l_j \) in \( \kappa \eta \mu \text{CPC} \);

\( nr_i := \) a (weak) normalizer of \( \phi_1(r_i) \) in \( \kappa \text{CPC} \);

\( nr_j := \) a (weak) normalizer of \( \phi_2(r_j) \) in \( \kappa \text{CPC} \);

if \( nr_i = nr_j \) then nothing needs to be done

else if \( nr_i > \kappa nr_j \) then \( f'_1 := \) arrow from \( nr_i \) to \( nr_j \); \( f := \) normalizer of \( f' \) in \( \kappa \eta \text{CPC} [f'] \)

else if \( nr_j > \kappa nr_i \) then \( f'_2 := \) arrow from \( nr_j \) to \( nr_i \); \( f := \) normalizer of \( f' \) in \( \kappa \eta \text{CPC} [f'] \)

else failure

\fi

\fi

\( P := P \cup \{f\}; \)

\( CN := \{(l_i, l_j) \mid (l_i, r_i), (l_j, r_j) \in P\}; \)

\[ \kappa \text{CPC} := \kappa \text{CPC} [f] \]

\fi

success

Note that the correctness of the above algorithm is given by Theorem 5.1: it only terminates if all critical normalizers have a reversed pushout (i.e. if the normalizers \( nr_i \) and \( nr_j \) resulting from the pushout diagram agree with each other), or if for some pair \( (nr_i, nr_j) \), no orientation for the new arrow can be determined. In the first case, Theorem 5.1 states that all elements of \( \kappa \text{CPC} \) have a normalizer (so the completion has finished successfully), while in the second case, the completion has failed.

6. Specification of the Categorical Data for the Knuth–Bendix Algorithm

The first of the CPC algorithms we will show to fit in our categorical model is the Knuth–Bendix completion procedure for term rewriting systems. For all categorical constructions used in the categorical CPC algorithm we will specify what they exactly correspond to in the Knuth–Bendix case. At the end of the section, we include an example to show how the categorical procedure follows the standard algorithm precisely.
Let an equational specification \((\Sigma, \mathcal{E})\) be given. Let \(T(\Sigma)\) be the corresponding term algebra, and let \(\succ_\kappa\) be a reduction ordering on \(T(\Sigma)\). The objects in \(\kappa\text{CPC}\) then are all elements of \(T(\Sigma)\). (We will denote all terms below by capital letters in order to conform to our notation for objects of a category.) The basic arrows between these are the oriented (w.r.t. \(\succ_\kappa\)) finitely many equations in \(\mathcal{E}\). We then extend the set of arrows of \(\kappa\text{CPC}\) as follows. An arrow between two terms \(T_1\) and \(T_2\) is a list of reduction steps leading from \(T_1\) to \(T_2\). A 1-step reduction is labeled \(\langle E, \sigma, C \rangle\) and has domain \(T_1 = C[\sigma(L)]\) and codomain \(T_2 = C[\sigma(R)]\), where \(E \in \mathcal{E}\) is defined as \(L \equiv R, L \succ R\). Any arrow is labeled by a list \([\langle E_1, \sigma_1, C_1 \rangle, \ldots, \langle E_n, \sigma_n, C_n \rangle]\). This list represents, in order, the reduction steps needed to reduce the domain to the codomain. Finally, all arrows with equal domain and codomain are identified.

**Proposition 6.1.** The objects and arrows of \(\kappa\text{CPC}\) defined on the basis of a set of equations \(\mathcal{E}\) in a specification \(\Sigma\) as outlined above form a category.

**Proof.** For any object we can define the identity arrow as an empty sequence of reduction steps. Composition of two arrows is defined by concatenating the lists of reduction steps corresponding to the two arrows. The identity laws and the associativity law follow immediately.

**Corollary 6.1.** \(\kappa-1\) and \(\kappa-2\) hold for the category \(\kappa\text{CPC}\) constructed above.

**Proof.** That the category \(\kappa\text{CPC}\) is a preorder (\(\kappa-1\)) follows from the identification of all arrows with equal domain and codomain; that all arrows respect \(\succ_\kappa\) (\(\kappa-2\)) is immediate from their construction.

We use the Dewey notation for terms. If a position \(k_1 \ldots k_n\) occurs in a term \(T\), then \(\pi_{T, (k_1 \ldots k_n)}\) maps \(T\) to \(T_{(k_1 \ldots k_n)}\).

**Corollary 6.2.** \(\pi-1\) and \(\pi-2\) hold for the category \(\kappa\text{CPC}\) constructed above.

6.2. THE CATEGORY OF MULTIPLIERS \(\mu\text{CPC}\)

In the context of term rewriting, the multipliers are given as substitutions of terms for variables. As in Stell (1992), the composition of substitutions is slightly non-standard. Normally, the composite of, say, \(\mu_1 = (f(y) \leftarrow x)\) (substituting \(x\) by \(f(y)\)) and \(\mu_2 = (z \leftarrow y)\) (substituting \(y\) by \(z\)) is \(\mu_2 \circ \mu_1 = (f(z) \leftarrow x)\). However, by the categorical composition laws forced upon us we have \(\mu_2 \circ \mu_1 = (f(z) \leftarrow x; z \leftarrow y)\) (simultaneous substitution of \(x\) by \(f(z)\) and \(y\) by \(z\)). Now consider any object in \(\kappa\text{CPC}\) defined by \(\Sigma\) and \(\mathcal{E}\), i.e. a term \(T \in T(\Sigma)\). Then, any substitution \(\sigma = (T_1 \leftarrow x_1, \ldots, T_n \leftarrow x_n)\) defines an arrow, labeled \(\sigma\), in \(\mu\text{CPC}\) from \(T\) to \(T[T_1 \leftarrow x_1, \ldots, T_n \leftarrow x_n]\). Arrows are identified by domain and label: \(\langle T, \sigma \rangle\).

**Proposition 6.2.** \(\mu\text{CPC}\) is a category.
The arrows of this operation of embedding, CPC equivalence class the same weight, the arrows of canonizer embedder $F$ a canonical representative for every term modulo CPC into $f$. Associativity of the substitutions is straightforward. □

We can view the multipliers, or arrows in $\mu_{\text{CPC}}$, as endofunctors on $\kappa_{\text{CPC}}$.

**Proposition 6.3.** Any term substitution $\sigma$ on $\Sigma$ is an endofunctor on the category $\kappa_{\text{CPC}}$ defined by $\Sigma$ and a finite set of equations $E$.

**Proof.** Let $\mu_\sigma$ be defined by mapping any object $T_1$ of $\kappa_{\text{CPC}}$ to $\sigma T_1$, and any arrow $f:T_1 \to T_2$ to $\sigma f: \sigma T_1 \to \sigma T_2$, the unique arrow between $\sigma T_1$ and $\sigma T_2$ in $\kappa_{\text{CPC}}$. We must verify that $\mu_\sigma$ is a functor. For any $T$ in $\kappa_{\text{CPC}}$, $\mu_\sigma(id_T) = id_{\mu_\sigma(T)}$. Also, for any $f:T_1 \to T_2$ and $g:T_2 \to T_3$, we have $\mu_\sigma(g \circ f) = \sigma(g \circ f) = (\sigma g) \circ (\sigma f) = \mu_\sigma(g) \circ \mu_\sigma(f)$. The middle identity follows from the definition of $\kappa_{\text{CPC}}$, the others from the definition of $\mu_\sigma$. □

Note that the functor $\mu_\sigma$ covers exactly all arrows labeled $\sigma$ in $\mu_{\text{CPC}}$. The additional structure imposed by the functor construction is the mapping between the arrows of $\kappa_{\text{CPC}}$—in other words, the functoriality of the multipliers expresses the compatibility of substitution with the reduction relation.

**Corollary 6.3.** The above specification of the multipliers satisfies all of $\mu$-1, $\mu$-2, $\mu$-3, and $\mu$-$4$ as required in Section 5.2.

**Proof.** The tree structure for terms guarantees $\mu$-1 and $\mu$-2; $\mu$-3 follows from the standard requirement that no variables occur on the right-hand side of a rewrite rule (pattern) if they do not occur on its left-hand side. Finally, the previous proposition guarantees $\mu$-4. □

### 6.3. Embedding at Contexts and Canonizing

The embedders are insertions of terms at given contexts. Since $T(\Sigma)$ is closed under this operation of embedding, CPC is equal to $\kappa_{\text{CPC}}$ and $\kappa$ is the identity functor. The arrows of $\kappa_{\text{CPC}}$ are endofunctors $\eta_{T,p}$ on $\text{CPC}$, where $T \in T(\Sigma)$ and $p$ a place in $T$.

The case of completion modulo equivalence relations $F$ is different. We assume that a canonical representative for every term modulo $F$ can be determined. Applying an embedder $\eta$ to a term $T$ in $\kappa_{\text{CPC}}$ may result in a term $\eta(T)$ not in canonical form; the canonizer $\kappa$ maps $\eta(T)$ from CPC to $[\eta(T)]$ in $\kappa_{\text{CPC}}$. If we assign all terms within an equivalence class the same weight, the arrows of CPC do not violate $\kappa$. (This is not always possible in other CPC algorithms.) Then, every embedder $\eta$ is a functor from $\kappa_{\text{CPC}}$ into CPC and $\kappa$ a functor from CPC into $\kappa_{\text{CPC}}$. Therefore, for every $\eta$, the composite $\kappa \eta$ is an endofunctor on $\kappa_{\text{CPC}}$.  


Corollary 6.4. The above specification of the embedders and canonization satisfies all of $\eta$-1, $\eta$-2, $\eta$-3, $\kappa$-3, $\kappa$-4, $\kappa$-5, $\kappa$-6 and $\kappa$-7 as required in Section 5.4.

Proof. In the case of ordinary term rewriting all but $\eta$-1 and $\eta$-2 follow from the fact that CPC and $\kappa$CPC are the same, while $\eta$-1 and $\eta$-2 follow immediately from the definition of embedding subterms into contexts. In the case of rewriting modulo some $F$ we require $\kappa$-3 from the start, and all of $\kappa$-4, $\kappa$-5, $\kappa$-6 and $\kappa$-7 follow from the construction of CPC. $\eta$-1 and $\eta$-2 follow as in ordinary rewriting, while $\eta$-3 follows from the compatibility of the reduction relation with the equivalence ($\kappa$-3). □

6.4. Normalizing Patterns and Finding Critical Pairs

For the Knuth–Bendix algorithm, unlike other CPC-algorithms, finding normalizers of arbitrary arrows in $\kappa$CPC is trivial. Recall that this is done by checking for normalizers in $\kappa\eta$CPC, whose objects correspond to the arrows in $\kappa$CPC. Since no non-trivial embedding will result in a smaller term w.r.t. $>\kappa$, it is clear that in the case of pure term rewriting, every arrow (of $\kappa$CPC) is its own normalizer. In the case of rewriting modulo equivalence relations, we may have to canonize the left- and right-hand sides of the pattern, but no multiplication or embedding will take place. Therefore, we have the following fact.

Observation 6.1. In the case of pure term rewriting, the normalizer (in $\kappa\eta$CPC) of any arrow $f: A \to B$ in $\kappa$CPC is $f$.

In the case of rewriting modulo equivalence relations, the normalizer of any arrow $f: A \to B$ in $\kappa\eta$CPC is $\kappa(f): \kappa(A) \to \kappa(B)$.

Finding critical pairs, or, categorically spoken, weak coproducts in $\kappa\eta\mu$CPC, amounts to finding equivalence preserving functors $\eta\mu$ for both objects (left-hand sides of patterns) under consideration. Let $A$ and $B$ be the objects for which a weak coproduct should be found in $\kappa\eta\mu$CPC. Assume that the corresponding patterns are $p_1: A \to A'$ and $p_2: B \to B'$ arrows in $\kappa$CPC.

Let us first consider ordinary term rewriting. Let $C$ fulfill $C = \eta_1(A)$ and $C = \eta_2(B)$, for distinct non-trivial embedders $\eta_1$ and $\eta_2$. Therefore, there are places $p$ and $q$, with $p \neq q$ and both non-zero, such that $C_p = A$ and $C_q = B$. If $p$ and $q$ share a non-empty initial sequence $r$, where $p = rp_1$ and $q = rq_1$, then $C$ is not a weak coproduct of $A$ and $B$ in $\kappa\eta\mu$CPC, as $\mu_r$ would also fulfill the corresponding cone conditions, and there is a non-identity arrow $\eta_{C,r}: C_r \to C$ in $\kappa\eta\mu$CPC. So, we may assume that $p$ and $q$ are disjoint places.

But in that case, the diagram in $\kappa$CPC resulting from $C, A$ and $B$ can easily be closed. First, reduce $C$ by the pattern $p_1$ at place $p$, obtaining $D = \eta_{C,p}(A')$, and then apply $p_2$ at $q$ in $D$: we obtain $F = \eta_{D,p}(B')$. Second, apply $p_2$ at $q$ in $C$, obtaining $E = \eta_{C,p}(B')$, and then $p_1$ at $p$ in $E$, obtaining $G = \eta_{E,p}(A')$. By checking all places we see that $F = G$. Since the application of a multiplier $\mu$ to either $A$ or $B$ before embedding is irrelevant to this argument, we conclude that we only have to look for weak coproducts in $\kappa\eta\mu$CPC such that one of the $\eta\mu$ is just a $\mu$, the corresponding $\eta$ being the identity functor. This corresponds to the normal check for critical overlaps in the Knuth–Bendix algorithm.

The above reasoning only holds because no canonization $\kappa$ has to be applied after embedding. Therefore, in the case of term rewriting modulo some $F$, we have to look for
all possible coproducts, including those formed by a pair \((\eta_1 \mu_1, \eta_2 \mu_2)\) with both \(\eta_1, \eta_2\) non-trivial. This corresponds to the use of extensions by Peterson and Stickel (1981).

The above can be summarized as follows.

**Observation 6.2.** In the case of pure term rewriting, all critical pairs are given by those weak coproducts \(C\) in \(\kappa \eta \mu \text{CPC}\) such that \(C = \eta_1 \mu_1(A) = \mu_2(B)\) or \(C = \mu_3(A) = \eta_2 \mu_4(B)\).

In the case of rewriting modulo equivalence relations, no restriction on the form of the coproducts can be made.

**Corollary 6.5.** In the case of completion of term rewriting systems all of \(\kappa \eta - 1\), \(\kappa \eta - 2\), and \(\kappa \eta \mu\) as required in Section 5.5 hold.

**Proof.** \(\kappa \eta - 1\) follows from the fact that embedding (also modulo equivalences) respects the ordering and can be composed; \(\kappa \eta - 2\) is an immediate consequence of the fact that embedding always results in a term greater than or equal to the original term, because of the (subsumption) ordering \(>\). So, the normalizer of an arbitrary arrow in \(\kappa \text{CPC}\) is always that arrow itself. \(\kappa \eta \mu\) follows from the intercomposability of the multipliers and embedders. \(\square\)

### 6.5. Example: Term Rewriting Modulo Equivalence Relations

**Example 6.1. (Free Commutative Groups)** The following example is taken from Peterson and Stickel (1981).

Let the following two group axioms be given:

\begin{align*}
e_1 : & \ 0 + x = x, \\
e_2 : & \ y + (-y) = 0. \\
\end{align*}

We derive two rewrite rules: \(r_1 : 0 + x \rightarrow x\) and \(r_2 : y + (-y) \rightarrow 0\) and work under the assumption that + is both associative and commutative. This means that in \(\text{CPC}\) there are additional arrows from associative and commutative variations of an object to a chosen standard representation; e.g. \((-y) + y = y + (-y)\) and back.

Note that we will use the complexity ordering on terms as defined by Peterson and Stickel (1981) based on the polynomial complexity measures by Lankford (1975a, b).

To form critical pairs, we first try overlaps on the rules as they are, and find the following superposition situation for \(r_1\) and \(r_2\): let \(\mu_1\) be \(\{x \leftarrow (-0), y \leftarrow 0\}\), then

\[0 + (-0)\]

so we obtain \(r_3 : -0 \rightarrow 0\).

Now, however, it is not possible to find any more critical pairs from the left-hand sides of the rules themselves, so we have to start looking for critical pairs modulo the equations defining the canonizer functor \(\kappa\) (i.e. associativity and commutativity).
Within \( \eta \text{CPC} \) we can construct several weak coproducts of the left-hand side of \( r_2 \) “with itself”. The first is: 

\[
\eta_{(-y)+b,<2>,(y)+y} = \eta_{b,+<1>,(y)+y} = \eta_2((-y) + y) \text{ where } \eta_2 = \{ y \rightarrow \langle -y \rangle \}.
\]

Denoting the first embedder here \( \eta_1 \) and the second \( \eta_2 \), we obtain, pictorially:

\[
\begin{align*}
-(-y) & + (-y) + y \\
r_2 \cdot \eta_1 & \quad r_2 \cdot \eta_2 \cdot \mu_2 \\
-(-y) & + 0 \\
r_1 \cdot \eta_1 & \quad 0 + y \\
-(-y) & \\
r_1 & \quad r_1 \\
y & \\
\end{align*}
\]

This gives rise to a fourth rule: \( r_4: -(-y) \rightarrow y \).

A second weak coproduct of the left-hand side of \( r_2 \) with itself within \( \eta \text{CPC} \) is given by \( \eta_3 = \eta_{b+(y),<1>} \) and \( \eta_4 = \eta_{(x+y)+y+b,<3>} \). As, for \( \mu_3 = \{ y \rightarrow \langle x + y \rangle \} \), we find \( \eta_3(\mu_3(y + (-y))) = \kappa \eta_4(y + (-y)) \).

This yields:

\[
\begin{align*}
-(-y) & + x + y + (-y) \\
r_2 \cdot \eta_2 \cdot \mu_3 & \quad r_2 \cdot \eta_4 \\
0 + (-y) & \quad -(x+y) + x + 0 \\\nr_1 & \quad r_1 \cdot \eta_1 \\
y & \quad -(x+y) + x \\
\end{align*}
\]

The fifth rule, obtained from this, is: \( r_5: -(x + y) + x \rightarrow -y \).

Due to the new rules, one more critical overlap occurs, namely between \( r_5 \) and \( r_2 \). Given \( \eta_5 = \eta_{b+(x),<1>} \) and \( \eta_6 = \eta_{-y+y+b} \), we find, for \( \mu_4 = \{ y \leftarrow \langle x + y \rangle \} \), we find \( \eta_6(\mu_4(y + (-y))) \), or, pictorially:

\[
\begin{align*}
-(x+y) & + x + (-x) \\
r_5 \cdot \eta_5 & \quad r_2 \cdot \eta_2 \cdot \mu_4 \cdot \kappa \\
(-y) & + (-x) \\
r_5 \cdot \eta_5 & \quad r_2 \cdot \eta_2 \cdot \mu_4 \cdot \kappa \\
-(x+y) & \\
\end{align*}
\]

This, finally, gives the rule \( r_6: -(x + y) \rightarrow (-y) + (-x) \).

With those six rules (where \( r_6 \) makes \( r_5 \) obsolete), the AC-reduction system for free commutative groups is complete.

7. Specification of the Categorical Data for the Gröbner Basis Algorithm

Next, we show that Buchberger’s Gröbner basis algorithm in polynomial ideal theory also fits in our categorical model. For all categorical constructions used in the categorical CPC algorithm we will specify what they exactly correspond to in the Gröbner basis case. Again, we conclude by an example showing how the categorical algorithm follows the traditional one precisely.
7.1. The reduction category $\kappa\text{CPC}$

Given a polynomial ring $K[x_1, \ldots, x_n]$ with field of coefficients $K$ and $n$ indeterminates, we define as the objects of $\kappa\text{CPC}$ in the Gröbner basis case all polynomials occurring in this ring, i.e. with coefficients in $K$ and in the indeterminates $x_1, \ldots, x_n$.

All objects are polynomials written in the standard canonical form, see below. In order to model multiplication by monomials analogously to substitution of terms for variables, we additionally have objects $X_0, X_1, \ldots, X_n$ representing dummy multiplication variables. These special objects are the variable objects. For the moment we only remark that the variable objects $X_i$ are introduced in order to be able to model multiplication properly in our model. Multiplication by monomials will be expressed as a substitution of variable objects by other terms. (All terms will turn out to contain these variable objects as subterms, see below.) A similar role will be played by the embedder object $Y$, which will be used for modeling embedding (or addition of polynomials), an operation which will lead us from $\kappa\text{CPC}$ into the (larger) category $\text{CPC}$.

As remarked above, all objects are polynomials in the standard canonical form. That is to say, given an order $>$ on the power products, the objects of $\kappa\text{CPC}$ are of the form
\[
c_1 x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} + \cdots + c_m x_0 x_1^{l_1} x_2^{l_2} \cdots x_n^{l_n} X_n + Y,
\]
where for every $1 \leq k < l \leq m$ we have $x_1^{i_1} \cdots x_n^{i_n} > x_1^{l_1} \cdots x_n^{l_n}$.

Again, the role of the $X_i$ and $Y$ will be explained fully in the sections on the multiplier and embedder functors, respectively.

The order $>$ on power products is extended to an ordering $>_{\kappa}$ on polynomials as follows. Let $LM(P)$ denote the leading monomial of a polynomial $P$ (the leading monomials are determined by $>$). Let $LPP(P)$ and $LC(P)$ denote the corresponding leading power product and leading coefficient, respectively. Let a total ordering $>_{\kappa}$ be given on the coefficient domain $K$. Then, we define $>_{\kappa}$ by stipulating that for any non-zero polynomial $P$, $P >_{\kappa} 0$, and for any two non-zero polynomials $P$ and $Q$, $P >_{\kappa} Q$ iff

(i) $LPP(P) > LPP(Q)$, or
(ii) $LC(P) >_{\kappa} LC(Q)$, if $LPP(P) = LPP(Q)$, or
(iii) $P - LM(P) >_{\kappa} Q - LM(Q)$ if $LM(P) = LM(Q)$ (this recursive definition works because we defined 0 as the smallest object above).

The arrows in the category $\kappa\text{CPC}$ for a given ideal $\mathcal{F}$ in the polynomial ring are constructed from basic patterns (derived from the polynomials in $\mathcal{F}$), multiplication by monomials, and addition of polynomials, as follows.

There is an arrow labeled $(F, M, Q)$ between the objects (polynomials) $P_1$ and $P_2$ if $P_1 = MF_1 + Q$ and $P_2 = MF_2 + Q$, where $F_1$ is the leading monomial of $F$ and $F_2 = F_1 - F$, for a polynomial $F \in \mathcal{F}$, and where $M \neq 0$ is a monomial.

All compositions of such arrows are also arrows of $\kappa\text{CPC}$, so an arbitrary arrow will be labeled by a finite (possibly empty) sequence of such labeled reduction steps, as $[\langle F_1, M_1, Q_1 \rangle, \ldots, \langle F_n, M_n, Q_n \rangle]$.

There are no arrows leading into or emerging from the variable objects $X_0, \ldots, X_n$, except for the trivial identity arrows.

As, in the context of completing reduction systems, we are interested mainly in the existence of arrows between given objects, and not so much in the various ways in which one object may be reduced to another, we identify all arrows between any two given
objects. Categorically spoken, every hom-set has at most one element, and the basic
reduction category $\kappa\text{CPC}$ is a preorder, as we required in the exposition in Section 5 on
the basic constituents of our categorical model.

**Proposition 7.1.** The objects and arrows of $\kappa\text{CPC}$ defined on the basis of a polynomial
ideal $\mathcal{F}$ in $K[x_1, \ldots, x_n]$ as outlined above form a category.

**Proof.** For any object $P$ we can define the identity arrow as an empty sequence of
reduction steps.

Composition of two arrows is also defined in a straightforward way: just concatenate
the lists of reduction steps corresponding to the two arrows.

The identity laws and the associativity law follow immediately. \(\square\)

**Corollary 7.1.** The requirements $\kappa$-1 and $\kappa$-2 hold for the category $\kappa\text{CPC}$ constructed
above.

**Proof.** That the category $\kappa\text{CPC}$ is a preorder ($\kappa$-1) follows immediately from all identifi-
cations of arrows with equal domain and codomain; that all arrows respect the ordering
$\succ_{\kappa}$ ($\kappa$-2) is immediately clear from the way the arrows are constructed and the ordering
is defined. \(\square\)

The precise specification of the places is connected to the tree representation of poly-
nomials we will use and now introduce.

**Definition 7.1.** (Tree Representation for Polynomials) For every polynomial

$$c_1X_0x_1^{i_1}X_1 \ldots x_n^{i_n}X_n + c_2X_0x_1^{i_2}X_1 \ldots x_n^{i_2n}X_n + \ldots + c_mX_0x_1^{i_m}X_1 \ldots x_n^{i_mv}X_n + Y$$

occurring as an object in $\kappa\text{CPC}$ we define a tree representation as follows.

The root node represents the addition function, all its branches represent the respective
monomials $c_jX_0x_1^{i_1}X_1 \ldots x_n^{i_jv}X_n$ (with a final branch, $Y$, in order to formulate addition
as a special embedding). Every branch representing the monomials consists of exactly
$n + 1$ subbranches. The first $n$ of these represent the power of $x_k$, $1 \leq k \leq n$, by being
a sequence (unary tree) of $i_{jk}$ $x_k$’s followed by the leave node $X_k$, where $X_k$ represents
the variable object for the indeterminate $x_k$. The last subbranch of a monomial consists
of a node with content $c_j$ followed by a leaf node $X_0$ where $X_0$ is the dummy variable to
be used for multiplication by coefficients.

The following picture illustrates the tree representation of polynomials. Note that the
sequences next to the arrows always indicate the path starting from the root position
(in this case representing the polynomial $3x_1^2x_2 + 7x_2 - 2x_1 + 300$), not just from the
previous object.
Now, given an arbitrary polynomial

\[ P = c_1 X_0 x_1^{i_1} X_1 \ldots x_n^{i_n} X_n + c_2 X_0 x_1^{i_2} X_1 \ldots x_n^{i_n} X_n + \cdots + c_m X_0 x_1^{i_m} X_1 \ldots x_n^{i_n} X_n + Y, \]

the place operators \( \pi_u \) on it are defined as follows:

(i) \( \pi_0(P) = \pi_0(P) = P; \)
(ii) \( \pi_{ij}(P) = c_j X_0 x_1^{i_1} X_1 \ldots x_n^{i_n} X_n \) for all \( 1 \leq j \leq m; \)
(iii) \( \pi_{ijk}(P) = x_k^{i_k} X_k \) for all \( 1 \leq j \leq m \) and \( 1 \leq k \leq n, \) provided \( i_k > 0; \) if \( i_k = 0, \)
\( \pi_{ijk}(P) = X_k; \)
(iv) \( \pi_{(n+1)j}(P) = c_j X_0 \) for all \( 1 \leq j \leq m; \)
(v) \( \pi_{(n+1)jk}(P) = x_k^{i_k} X_k, \) where \( 1^r \) indicates a sequence of \( r \) copies of \( 1, \) for all \( 1 \leq j \leq m \) and \( 1 \leq k \leq n, \) provided \( i_k - r > 0; \) if \( i_k - r = 0, \)
\( \pi_{(n+1)jk}(P) = X_k; \)
(vi) \( \pi_{(n+1)(n+1)}(P) = X_0; \)
(vii) \( \pi_{((n+1)(n+1))j}(P) = Y; \)
(viii) no other \( \pi_u \) are defined on \( P \) (due to the structure of polynomials, all places have one of the forms as in (i)--(vii)).

**Corollary 7.2.** \( \pi-1 \) and \( \pi-2 \) hold for the category \( \kappa CPC \) constructed above.

### 7.2. The Category of Multipliers \( \mu CPC \)

The multipliers represent multiplication by monomials. The category \( \mu CPC \) has the same objects as \( \kappa CPC \) itself, and there is an arrow with domain \( P_1 \) and codomain \( P_2 \) if there is a monomial \( M \in K[x_1, \ldots, x_n] \) such that \( MP_1 = P_2. \)

Note that \( P_2 \) will be in canonical form again. To be precise, for

\[ P_1 = c_1 X_0 x_1^{i_1} X_1 \ldots x_n^{i_n} X_n + c_2 X_0 x_1^{i_2} X_1 \ldots x_n^{i_n} X_n + \cdots + c_r X_0 x_1^{i_r} X_1 \ldots x_n^{i_n} X_n + Y, \]

where for every \( k, l, 1 \leq k < l \leq r \) we have \( x_1^{i_k} \ldots x_n^{i_k} > x_1^{i_l} \ldots x_n^{i_l} \) and for

\[ M = m X_0 x_1^{i_1} X_1 x_2^{j_2} X_2 \ldots x_n^{j_n} X_n + Y, \]

we have

\[
    P_2 = mc_1 X_0 x_1^{i_1+j_1} X_1 \ldots x_n^{i_n+j_n} X_n + mc_2 X_0 x_1^{i_2+j_1} X_1 \ldots x_n^{i_n+j_n} X_n + \cdots + mc_r X_0 x_1^{i_r+j_1} X_1 \ldots x_n^{i_n+j_n} X_n + Y,
\]
where trivially for every \( k, l, \) and \( 1 \leq k < l \leq r \) we have \( x_1^{k_1+j_1} \ldots x_n^{k_n+j_n} > x_1^{l_1+j_1} \ldots x_n^{l_n+j_n} \).

This arrow will be labeled \( M : P_1 \to P_2 = MP_1 \). (Note that this labeling is not unique; there will be many arrows labeled by the same monomial \( M \), but all with a different domain. In order to give a unique name to every arrow, we have to denote them by both their domain and their label, \( \langle P_1, M \rangle \).) The effect of \( M \) can be described as the composition of \( n+1 \) substitutions, namely \( (mX_0 \leftarrow X_0, x_1^{j_1}X_1 \leftarrow X_1, \ldots, x_n^{j_n} X_n \leftarrow X_n) \).

**Proposition 7.2.** \( \mu_{\text{CPC}} \) is a category.

**Proof.** All identity arrows trivially exist (labeled \( 1 \)) and for any two arrows \( \langle P_1, M_1 \rangle \) and \( \langle P_2, M_2 \rangle \) such that the codomain of the first is \( P_2 \), we can form the composite arrow \( \langle P_1, (M_2M_1) \rangle \). Obviously, \( M_2M_1 \) will again be a monomial, and the identity and associativity laws follow immediately from the corresponding properties of the polynomial ring \( K[x_1, \ldots, x_n] \).

As explained in Section 5, the multipliers do not only serve as arrows in the category \( \mu_{\text{CPC}} \) but also as endofunctors on \( \kappa_{\text{CPC}} \). Concretely, this looks as follows in the case of Gröbner bases.

**Proposition 7.3.** Any monomial \( M \in K[x_1, \ldots, x_n] \) is an endofunctor on the category \( \kappa_{\text{CPC}} \) defined by \( K[x_1, \ldots, x_n] \) and any ideal \( \mathcal{F} \).

**Proof.** The functor \( \mu_M \) defined by the monomial \( M \) operates as follows on \( \kappa_{\text{CPC}} \): any object \( P_1 \) is mapped to \( MP_1 \), while any arrow \( f : P_1 \to P_2 \) is mapped to \( Mf : MP_1 \to MP_2 \), the unique arrow in \( \kappa_{\text{CPC}} \) from \( MP_1 \) to \( MP_2 \).

We have to verify that all functor properties are fulfilled by this operation.

For any object \( P \) in \( \kappa_{\text{CPC}} \), \( \mu_M(\text{id}_P) = \text{id}_{MP} \). Also, for any arrows \( f : P_1 \to P_2 \) and \( g : P_2 \to P_3 \), we have \( \mu_M(g \circ f) = \mu_M(g) \circ \mu_M(f) \). The middle identity is true by the definition of arrow composition in \( \kappa_{\text{CPC}} \), the others by definition of \( \mu_M \). \( \square \)

Note that the functor \( \mu_M \) exactly covers all arrows labeled \( M \) in \( \mu_{\text{CPC}} \). The extra structure imposed by the functor construction is the mapping between the arrows of \( \kappa_{\text{CPC}} \)—in other words, the functoriality of the multipliers expresses the compatibility of the multiplication by monomials with the reduction.

**Corollary 7.3.** The above specification of the multipliers satisfies all of \( \mu-1, \mu-2, \mu-3 \) and \( \mu-4 \).

**Proof.** Checking that \( \mu-1 \) and \( \mu-2 \) hold is straightforward from looking at the tree structure defined for polynomials. Since \( \mu_1(A) \) can only be equal to \( \mu_2(A) \) if the corresponding monomials are the same, \( \mu-3 \) is immediate. Finally, the previous proposition guarantees \( \mu-4 \). \( \square \)
7.3. Embedding at Contexts and Canonizing

Intuitively, the embedders here are given by addition of polynomials. The objects of the category of embedders, \( \eta \text{CPC} \), and those of \( \text{CPC} \) are sums of monomials, without any restriction on the ordering of the power products (the same power product may in fact occur in several subterms).

So, a typical object of these two categories is of the form

\[
c_1 x_0^{i_1} x_1 \ldots x_n^{i_n} x_n + c_2 x_0^{i_1} x_1 \ldots x_n^{i_2} x_n + \cdots + c_m x_0^{i_m} x_1 \ldots x_n^{i_m} x_n + Y,
\]

for coefficients \( c_i \in K \).

The arrows of \( \eta \text{CPC} \) are defined as follows. Given an arbitrary polynomial \( P \), and an arbitrary object \( Q \) of \( \eta \text{CPC} \), there is an arrow \( P;Q \rightarrow Q + P \), where \( Q + P \) is the concatenation of \( Q \) and \( P \) (modulo canonization). To be precise, for

\[
Q = c_1 x_0^{i_1} x_1 \ldots x_n^{i_n} x_n + c_2 x_0^{i_1} x_1 \ldots x_n^{i_2} x_n + \cdots + c_m x_0^{i_m} x_1 \ldots x_n^{i_m} x_n + Y
\]

and

\[
P = d_1 x_0^{i_1} x_1 \ldots x_n^{i_n} x_n + d_2 x_0^{i_1} x_1 \ldots x_n^{i_2} x_n + \cdots + d_l x_0^{i_l} x_1 \ldots x_n^{i_l} x_n + Y.
\]

In the tree representation, embedding a polynomial \( P \) into \( Q \) corresponds to substituting the occurrence of \( Y \) in \( Q \) by \( P \). This results in a non-canonical representation of the polynomial, not just because the ordering of the power products is not necessarily obeyed, but also because the polynomial \( P \) occurs at the same level in the tree (one below the root) as the monomials in \( Q \).

The canonizer functor \( \kappa \) transforms the polynomial \( Q + P \) (or \( \eta_{Q,(l(n+1))}(P) \)) into the corresponding canonical form.

Compare the above construction of \( \eta \text{CPC} \) with the construction of the category of multipliers \( \mu \text{CPC} \). Again, there will be infinitely many arrows labeled by any \( P \), so to specify an arrow completely we have to equip it with both domain and label, \( (P,Q) \).

**Proposition 7.4.** \( \eta \text{CPC} \) is a category.

**Proof.** All identity arrows trivially exist (labeled by the empty polynomial, which we will denote by 0) and for any two arrows \( (P_1, Q_1) \) and \( (P_2, Q_2) \) such that the codomain of the first is \( P_2 \), we can form the composite arrow \( (P_1, (Q_1 + Q_2)) \). Obviously, \( Q_1 + Q_2 \) will again be a polynomial, and the identity and associativity laws follow immediately. \( \square \)

With the help of the arrows of \( \eta \text{CPC} \) we can define the embedder functors \( \eta_R : \kappa \text{CPC} \rightarrow \text{CPC} \). For every object \( P \) of \( \kappa \text{CPC} \), the image under \( \eta_R \) is \( \eta_R(P) = P + R \). For every arrow (reduction) \( (F,M,Q) : P_1 \rightarrow P_2 \) in \( \kappa \text{CPC} \), \( \eta_R((F,M,Q)) : \eta_R(P_1) \rightarrow \eta_R(P_2) \) is defined in \( \text{CPC} \) as \( (F,M,Q + R) \). Moreover, for any two polynomials \( P, Q \) with the same canonical representation in \( \text{CPC} \) we add arrows \( e_{P,Q} : P \rightarrow Q \) and \( e_{Q,P} : Q \rightarrow P \) in \( \text{CPC} \). By imposing closure under composition of arrows we ensure that \( \text{CPC} \) is a category.
Proposition 7.5. For every polynomial \( P \) that is an object of \( \kappa \text{CPC} \), the embedder \( \eta_P \) is a functor from \( \kappa \text{CPC} \) into \( \text{CPC} \).

Proof. That \( \eta_P \) maps objects \( Q \) of \( \kappa \text{CPC} \) to objects in \( \text{CPC} \) is clear.

Let an arrow \( (F, M, Q) \) in \( \kappa \text{CPC} \) with domain \( P_1 = MF_1 + Q \) and codomain \( P_2 = MF_2 + Q \) be given. Then \( \eta_P((F, M, Q)): \eta_P(P_1) \to \eta_P(P_2) \) is defined as the arrow \( (F, M, \eta_P(Q)) \).

It is readily checked that the required functor properties are fulfilled by spelling out the objects (polynomials) term by term.

Corollary 7.4. The above specification of the embedders and canonization satisfies all of \( \eta^{-1} \), \( \eta^{-2} \), \( \eta^{-3} \), \( \kappa^{-4} \), \( \kappa^{-5} \), \( \kappa^{-6} \) and \( \kappa^{-7} \).

Proof. First, \( \kappa^{-3} \) follows from the semi-compatibility of the reduction relation with embedding into contexts. All of \( \kappa^{-4} \), \( \kappa^{-5} \), \( \kappa^{-6} \) and \( \kappa^{-7} \) follow immediately from the construction of \( \text{CPC} \). It is easy to check \( \eta^{-1} \) and \( \eta^{-2} \) by means of the chosen tree representation for polynomials, and the previous proposition guarantees \( \eta^{-3} \).

Using embedders, the canonizer \( \kappa \), and the order relation, we finally define the category \( \kappa \eta \text{CPC} \) with as objects the arrows of \( \kappa \text{CPC} \).

The category \( \kappa \eta \text{CPC} \) has as objects the reductions in the universe (i.e. all arrows of \( \kappa \text{CPC} \)) and as arrows all non-decreasing embedders: for arrows \( \alpha: A_1 \to A_2 \) and \( \beta: B_1 \to B_2 \) in \( \kappa \text{CPC} \) we have an arrow between the objects \( \alpha \) and \( \beta \) in \( \kappa \eta \text{CPC} \) labeled \( C \) (\( C \) an object of \( \kappa \text{CPC} \)) iff:

(i) \( \kappa \eta \text{C}(A_1) = B_1 \) and \( \kappa \eta \text{C}(A_2) = B_2 \); this condition will hereafter be abbreviated as \( \kappa \eta \text{C}(\alpha) = \beta \);

(ii) \( A_1 \geq_{\kappa} A_2, B_1 \geq_{\kappa} B_2 \) (these two inequalities follow from the fact that we are dealing with arrows in \( \kappa \text{CPC} \)) and \( A_1 \geq_{\kappa} B_1 \).

Again, to name an arrow, we have to give both its domain and its label \( \langle \alpha, C \rangle \).

Proposition 7.6. \( \kappa \eta \text{CPC} \) is a category.

Proof. The identity arrows are the non-decreasing embedders labeled 0. Given arrows \( \langle \alpha, C_1 \rangle \) and \( \langle \beta, C_2 \rangle \) such that \( \kappa \eta \text{C}(\alpha) = \beta \), their composition is defined as \( \langle \alpha, \kappa(C_1+C_2) \rangle \).

The identity and associativity laws are now straightforward to check.

7.4. Normalizing Patterns and Finding Critical Pairs

We conclude the discussion of the incorporation of the Gröbner basis algorithm in our framework by making some observations on what constructing patterns (or, categorically, by normalizing in \( \kappa \eta \text{CPC} \)) and critical pairs (by constructing weak coproducts in \( \kappa \eta \mu \text{CPC} \)) amounts to for the individual algorithms.

Finding normalizers of arbitrary arrows always results in having the leading monomial on the left-hand side and the remainder on the right-hand side (compare the remarks on the ordering \( >_{\kappa} \) in this case).
Observation 7.1. In the case of constructing Gröbner bases, all normalizers are of the form \( M \rightarrow P \) (an arrow in \( \kappa \cpc \) and an object in \( \kappa \mu \cpc \)), where \( M \) is a monomial such that \( M >_{\kappa} P \).

Because of this, constructing weak coproducts of left-hand sides of patterns always amounts to constructing such coproducts for monomials (instead of arbitrary polynomials). Given two monomials \( M_1 \) and \( M_2 \), it follows immediately that there are only two ways to do that: either by using the embedders \( \eta_{M_2}, (2) \) and \( \eta_{M_1}, (2) \), which applied to \( M_1 \) and \( M_2 \), respectively, result in \( M_1 + M_2 \) (modulo canonizing by means of \( \kappa \)), or by using multipliers \( N_1 \) and \( N_2 \) such that \( M_1 N_1 = M_2 N_2 \) and no smaller common multiple can be found. The first case gives rise to a trivial confluence, while the second amounts to constructing the least common multiple of \( M_1 \) and \( M_2 \). As the first case is clearly irrelevant for the algorithm, we therefore may as well restrict ourselves to constructing weak coproducts within the subcategory of \( \kappa \mu \cpc \) that has only monomials as its objects. But then it is immediately clear that all occurring embedders are the trivial ones, \( \eta_A,() \). Therefore, we can restrict our attention to finding weak coproducts by the multipliers alone, and since these are endofunctors on \( \kappa \cpc \) that means that looking for weak coproducts in \( \kappa \mu \cpc \) simplifies to looking for weak coproducts in \( \mu \cpc \).

Observation 7.2. In the case of the construction of Gröbner bases, all critical pairs are given by those weak coproducts \( C \) in \( \kappa \mu \cpc \) such that \( C = \mu_1(A) = \mu_2(B) \), i.e. by the weak coproducts of \( \mu \cpc \).

Note, moreover, that in the case of Gröbner basis construction, the coproducts in \( \mu \cpc \) are unique (and correspond to the least common multiple of the monomials involved). This also implies that one does not need to look for coproducts of the left-hand side of one and the same pattern (since this would just be that left-hand side, and the two reductions arising from it would always be equal).

Corollary 7.5. In the case of Gröbner basis construction all of \( \kappa \eta - 1 \), \( \kappa \eta - 2 \) and \( \kappa \mu \) hold.

Proof. \( \kappa \eta - 1 \) follows from the fact that we can compose arbitrary arrows in \( \kappa \eta \cpc \) (with appropriate domain and codomain) because of the transitivity of the ordering relation \( >_{\kappa} \) (on left-hand sides, or domains). \( \kappa \eta - 2 \) follows from the fact that the ordering on the canonized polynomials is total, and \( \kappa \mu \) from the composability (also among each other) of the multipliers and embedders. □

7.5. Example: Gröbner basis

Example 7.1. (Petri net) Let us look at the following example from Buchberger [1983b], where the following three polynomials are used to describe the transitions of a reversible Petri net.

Input polynomials:

\[
\begin{align*}
F_1 &= as - c^2 s \\
F_2 &= bs - cs \\
F_3 &= s - f.
\end{align*}
\]
We recall that we use the total degree ordering with $s > f > c > b > a$ and that $\mathbb{Q}$ is our coefficient domain. (The ordering $>_K$ on $\mathbb{Q}$ is defined as: $n >_K m$ if either $|n| > |m|$ or $0 > n = -m$. So, the absolute values are compared first, and a negative number is taken to be greater, w.r.t. $>_K$, than its absolute value.)

In this context, the category $\kappa CPC$ has as object all elements (in canonical form) of $\mathbb{Q}[a, b, c, f, s]$. The arrows in $\kappa CPC$ are all those that can be obtained as compositions of canonified embeddings of multiplied instances of $F_i$, $i \in \{1, 2, 3\}$.

The category $\kappa\eta CPC$, from which we will obtain the canonified form of the patterns, has as objects all arrows of $\kappa CPC$ and as arrows all non-decreasing embedders between them. Taking, for example, the object $s - f \rightarrow 0$ (corresponding to $F_3$), we obtain as normalizer the object $s \rightarrow f$, the result of applying the embedder $\eta_f$ and canonizing.

From the polynomials $F_1, F_2$, and $F_3$, we obtain the following three patterns by embedding and canonizing (that is, normalizing in $\kappa\eta CPC$):

- $r_1 = c^2 s \rightarrow as$
- $r_2 = cs \rightarrow bs$
- $r_3 = s \rightarrow f$

In the first iteration, we construct the weak coproduct of the left-hand sides of $r_2$ and $r_3$ in $\kappa\eta\mu CPC$. Recall that $\kappa\eta\mu CPC$ has the same objects as $\kappa CPC$ (so, here, those representing the elements of $\mathbb{Q}[a, b, c, f, s]$) and as arrows compositions of a multiplier followed by an embedder (where one or both may be the identity). In the case of Gröbner bases we can simply construct these coproducts in $\mu CPC$, which has the same objects as $\kappa\eta\mu CPC$ but only multipliers (here multiplications by monomials in $\mathbb{Q}[a, b, c, f, s]$) as arrows. We therefore find $\mu_1 = 1$ and $\mu_2 = c$.

The critical normalizer of the arrow $cf \rightarrow bf$ in $\kappa\eta CPC$ is simply that arrow itself so we obtain:

- $r_4 = cf \rightarrow bf$

(That the arrow $cf \rightarrow bf$ is indeed its own normalizer in $\kappa\eta CPC$ follows from the fact that no addition of a polynomial in $\mathbb{Q}[a, b, c, f, s]$ to both left- and right-hand side leads to an arrow with a smaller left-hand side while preserving the property that the left-hand side is greater than the right-hand side.)

In the second iteration, we take $\mu_1 = 1$ and $\mu_2 = c$ and construct the weak coproduct of the left-hand sides of $r_1$ and $r_2$ in $\mu CPC$: 

- $r_4 = cf \rightarrow bf$
The critical normalizer of the arrow $b^2 f \to af$ is precisely that arrow, so we add:

$$r_5 = b^2 f \to af.$$ 

We now find that in the category $\kappa CPC\left[r_4, r_5\right]$ all the weak coproducts of the patterns in $\mu CPC$ (i.e. in $\kappa \eta \mu CPC$) have reversed pushouts. As an example, we check $r_1$ and $r_3$, with the multipliers $\mu_1 = 1$ and $\mu_2 = c^2$.

### 8. Specification of the Categorical Data for the Resolution Algorithm

The last CPC algorithm we will show that fits into our categorical model is Robinson’s resolution algorithm for proving the unsatisfiability of (the negation of) statements, including its extension to (finite) many-valued logics due to Baaz. For all categorical constructions used in the categorical CPC algorithm we will specify what they exactly correspond to in the resolution case. Again, we conclude with an example of the categorical algorithm in the case of resolution, which will indicate where the categorical formulation differs slightly from the traditional algorithm.

#### 8.1. The Reduction Category $\kappa CPC$

We will first discuss the case of resolution in classical first-order logic.

Let a logical language $\mathcal{L}$ and a finite set of clauses $\mathcal{C}$ in that language $\mathcal{L}$ be given. Note that a clause is nothing but a set of literals (positive or negative (negated) in the
classical case; equipped with a truth value when generalizing to the case of many-valued
logics). The semantical interpretation of a clause is simply the disjunction of the literals
occurring in it.

The objects of the category $\kappa\text{CPC}$ defined by $\mathcal{L}$ are simply all clauses that can be
formed within $\mathcal{L}$. All clauses are supposed to be in canonical form, i.e. with no literal
appearing more than once and such that any tautology is written as $\{\top\}$.

The arrows are derived from the clauses in $\mathcal{C}$ as follows. For every clause $C$ in $\mathcal{C}$, where
$C = \{L_1, \ldots, L_n\}$, we form all arrows that are created by taking an arbitrary literal $L_i$
out of $C$, where $C$ is any clause in $\mathcal{C}$, and constructing the arrow with domain $\{\neg L_i\}$
and codomain $C'$, where $C'$ is $C \setminus \{L_i\}$. So, all these arrows have a single-literal clause
as their domain. These arrows, labeled as $C_L; \{\neg L\} \to C \setminus \{L\}$, are the basic arrows or
patterns.

In order to justify this from the ordinary interpretation of resolution, consider the
following.

A clause $C = \{L_1, \ldots, L_n\}$ normally corresponds to the assertion that the disjunction
of its literals is true, which we will write $\top \iff \bigvee_{i=1}^n L_i$. Logically, this is equivalent to
$\neg L_i \iff \bigwedge_{j \neq i} L_j$, for all $j \in \{1, \ldots, n\}$. Doing this for all clauses gives the $n$
patterns described above.

Both domain and codomain of an arrow are conceived as the disjunction of the literals
occurring in the corresponding clause.

From the basic patterns, we can obtain derived patterns by adding the same literal on
both sides. This will be described precisely when embedding is discussed, but one special
(and essential) case should be mentioned here.

Given an arrow $\{L\} \to C$, we can add the literal $\neg L$ on both sides to obtain $\{L, \neg L\} \to
C \cup \{\neg L\}$. The canonizer operation (to be defined when embedding is discussed) transforms $\{L, \neg L\}$ to simply $\{\top\}$.

Considering the arrows to indicate derivability, it is clear that with respect to this
reduction we will have to consider $\{\bot\}$ (the set containing only the empty clause, traditionally denoted $\square$ in the literature on automated deduction) as the initial object (from falsity we can derive anything) and $\{\top\}$ as the terminal object (it follows from every statement). The goal of the resolution procedure is to construct an arrow from $\{\top\}$ to $\{\bot\}$, to showing unsatisfiability. In categorical terms, this amounts to showing that the terminal and initial object are isomorphic (a so-called zero object). The technique we use here for doing that is the construction of a category based on the original set of clauses, as specified in the categorical CPC algorithm.

The ordering $\succ_\kappa$ on $\kappa\text{CPC}$ will only be a preorder, so it can happen that $A \succ_\kappa B$ and
$B \succ_\kappa A$ for different objects $A, B$ (and, indeed, $A \succ_\kappa A$ will hold for all objects $A$). We
only put the following restrictions on $\succ_\kappa$:

(i) $D \succ_\kappa C$ if $C$ has only one element (this technical demand will be necessary to
obtain the proper normalizers in the auxiliary category $\kappa\eta\text{CPC}$ later on);
(ii) if $\sigma(A) \subseteq B$ for two arbitrary clauses $A, B$ and an arbitrary substitution (multiplier)
$\sigma$, then $A \succ_\kappa B$ (this reflects the fact that the truth of $A$ implies $\sigma(A)$, which in
turn implies the truth of $B$; note, by the way, that $A \equiv_\kappa B$ is defined as equality
on sets); as a special case of this we have, for all $A$ and $\sigma$, $A \succ_\kappa \sigma(A)$;
(iii) as remarked before, for any clause $A$, $\{\bot\} \succ_\kappa A \succ_\kappa \{\top\}$.

By the above restrictions we have enforced that $\succ_\kappa$ respects logical consequence.
We will see below that we can define some elementary operations (like the addition of clauses to both sides, or substitution), and we can compose the arrows generated by the above definition of the reduction.

Arrows corresponding to one reduction step are labeled by \( \langle D_1, \sigma, C_L \rangle \), where \( D_1 \) is the domain of the arrow, that is a set of literals, which, moreover, contains \( \sigma(\neg L) \), and the codomain \( D_2 = (D_1 \cup \sigma C) - \{ \sigma L \} \). Arbitrary arrows are labeled by lists of sequences \([\langle D_1, \sigma_1, C_{L_1} \rangle, \ldots, \langle D_n, \sigma_n, C_{L_n} \rangle]\).

Composition is defined by concatenation. In the case of two arrows \( \langle D_1, \sigma_1, C_L \rangle \) and \( \langle D_2, \sigma_2, C'_L \rangle \) such that \( D_2 \) is equal to the codomain of the first arrow \( (D_1 \cup \sigma C) - \{ \sigma L \} \), the composition is the sequence \( \langle D_1, \sigma_1, C_L \rangle \langle D_2, \sigma_2, C'_L \rangle \) with domain \( D_1 \) and codomain
\[D_3 = (D_2 \cup \sigma_2 C') \setminus \{ \sigma_2 L' \} = ((D_1 \cup \sigma_1 C) \setminus \{ \sigma_1 L \}) \cup \sigma_2 C' \setminus \{ \sigma_2 L' \}.
\]

Again, we identify all arrows with equal domain and codomain, thereby obtaining a preorder. Finally, we require that \( \bot \) is an initial and \( \top \) is a terminal object and add the relevant arrows (in particular \( \{ \bot \} \rightarrow \{ \top \} \)).

For the many-valued case, the construction is slightly different. Let \( W = \{ w_1, \ldots, w_n \} \) be the set of all truth values, equipped with an ordering \( \succ_W \). Let \( \top \) be the maximal and \( \bot \) be the minimal truth value in \( W \) w.r.t. \( \succ_W \). (Note that as a matter of fact we only require that there is a maximal and a minimal element w.r.t. \( \succ_W \), and not that \( \succ_W \) is total.)

Again, we construct a category \( \kappa \text{CPC} \) with \( \{ \bot \} \) as the initial and \( \{ \top \} \) as the terminal object, and the resolution algorithm tries to show that they are isomorphic.

We construct a set of patterns out of a given set of clauses \( \mathcal{C} \) as follows.

Every non-empty clause \( C = \{ L_1^{n_1} \ldots L_m^{n_m} \} \) is interpreted as \( \top \leftarrow C \). From this we obtain \( m(n-1) \) basic patterns, namely for all \( i = 1, \ldots, m : \{ L_i \} \leftarrow \{ L_i^{n_i} \} \), for every \( w \neq w_i \) (giving \( n-1 \) patterns for each \( L_i \)).

In this case, the canonizer \( \kappa \) will transform \( \{ L_i^{w_i} | w_i \in W \} \) to \( \{ \top \} \), for any literal \( L \). Finally, we again require that \( \{ \bot \} \) is an initial and \( \{ \top \} \) is a terminal object and add the relevant arrows.

**Proposition 8.1.** The objects and arrows of \( \kappa \text{CPC} \) defined on the basis of a set of clauses \( \mathcal{C} \) in a logical language \( \mathcal{L} \) as outlined above form a category.

**Proof.** Exactly as in the case of Gröbner bases. \( \square \)

**Corollary 8.1.** The requirements \( \kappa-1 \) and \( \kappa-2 \) hold for the category \( \kappa \text{CPC} \) constructed above.

**Proof.** That the category \( \kappa \text{CPC} \) is a preorder \( (\kappa-1) \) follows immediately from all identifications of arrows with equal domain and codomain; that all arrows respect the ordering \( \succ_k \) \( (\kappa-2) \) is immediately clear from the way the ordering is defined. (The fact that \( \kappa-1 \) holds implies in this case that all objects that can be considered as logical consequences from each other are isomorphic: suppose \( f : A \rightarrow B \) and \( g : B \rightarrow A \), then \( f \circ g = 1_B \) and \( g \circ f = 1_A \), which corresponds to the intuition that, logically, they are equivalent.) \( \square \)

Again, the places are based on a tree representation of the objects, just like in the case of Gröbner bases. We now introduce this tree representation.
Definition 8.1. (Tree Representation for Clauses) For every clause $C = \{L_1^{w_1}, \ldots, L_n^{w_n}\}$, we define a tree representation with a root node containing the $n$-ary disjunction symbol and as its branches all the individual literals $L_i^{w_i}$, which have their positions represented by the Dewey notation introduced above for the case of term rewriting. In addition, for any clause $C = \{L_1^{w_1}, \ldots, L_n^{w_n}\}$ an extra position $C_{(n+1)}$ is defined, and occupied by a constant $Y$; the role of this $Y$ will be explained in the discussion of embedders.

More specifically, given an arbitrary clause $C = \{L_1^{w_1}, \ldots, L_n^{w_n}\}$, the place operators $\pi_n$ on it are defined as follows:

\[(i)\) $\pi_0(C) = \pi_{(0)}(C) = C$;
\[(ii)\) $\pi_{(j)}(C) = L_j^{w_j}$ for all $1 \leq j \leq n$;
\[(iii)\) $\pi_{(j)}(C) = (L_j^{w_j})_{(k)}$ for all $1 \leq j \leq n$ and $\langle k \rangle$ occurring as place in $L_j^{w_j}$;
\[(iv)\) $\pi_{((n+1))}(C) = Y$;
\[(v)\) no other $\pi_n$ are defined on $C$, as all places occurring in $C$ are of the form covered in (i)-(iv).

Corollary 8.2. $\pi$-1 and $\pi$-2 hold for the category $\kappa CPC$ constructed above.

8.2. The category of multipliers $\mu CPC$

In the construction of $\mu CPC$ and that of the endofunctors $\mu_\Sigma$ on $\kappa CPC$, there is no difference whatsoever between the resolution case and the term rewriting case, except in the domain of the terms (here the language $L$ instead of the signature $\Sigma$). The construction works in exactly the same way. We will, nevertheless, repeat the argument, for the sake of establishing all notions independently.

In the context of resolution, the multipliers are given by all substitutions of terms for variables. In order to define $\mu CPC$, consider any object in the category $\kappa CPC$ defined by $L$ (for any finite set of clauses $C$). This is a clause $C$. A substitution $\sigma = (T_1 \leftarrow x_1, \ldots, T_n \leftarrow x_n)$ defines an arrow (labeled by $\sigma$) in $\mu CPC$ with domain $C$ and codomain $\sigma(C) = C[T_1 \leftarrow x_1, \ldots, T_n \leftarrow x_n]$. Again, to actually name an arrow, we have to give both its domain and its label: $\langle C, \sigma \rangle$.

Proposition 8.2. $\mu CPC$ is a category.

Proof. Again, all identity arrows trivially exist (the empty substitutions, which we will label $\sigma_0$ from now on). Moreover, for any two arrows $\langle C_1, \sigma_1 \rangle$ and $\langle C_2, \sigma_2 \rangle$ such that the codomain of the first is $C_2$, we can form the composite arrow $\langle C_1, \sigma_2 \sigma_1 \rangle$. Here, the composite substitution $\sigma_2 \sigma_1$ is defined as follows. Let $\sigma_1 = (U_1 \leftarrow x_1, \ldots, U_n \leftarrow x_n)$ and $\sigma_2 = (W_1 \leftarrow x_1, \ldots, W_n \leftarrow x_n)$. (Of course, normally some of the $U_i$ and $W_i$ will just be equal to $x_i$.) Then $\sigma_2 \sigma_1 = (\sigma_2(U_1) \leftarrow x_1, \ldots, \sigma_2(U_n) \leftarrow x_n)$.

Now, the identity laws hold trivially, while checking the associativity of the substitutions is a straightforward exercise. $\square$

Again, we can view the multipliers, or arrows in $\mu CPC$, as endofunctors on $\kappa CPC$. The construction is analogous with that in the Gröbner basis case.
Proposition 8.3. Any term substitution $\sigma$ on $L$ is an endofunctor on the category $\kappa\mathrm{CPC}$ defined by $L$ and any finite set of clauses $C$.

Proof. The functor $\mu_\sigma$ defined by the substitution $\sigma$ operates as follows on $\kappa\mathrm{CPC}$: any object $C_1$ is mapped to $\sigma C_1$, while any arrow $f: C_1 \to C_2$ is mapped to $\sigma f: \sigma C_1 \to \sigma C_2$, the unique arrow in $\kappa\mathrm{CPC}$ from $\sigma C_1$ to $\sigma C_2$.

The verification that all functor properties are fulfilled is analogous with the case of term rewriting systems. $\square$

Again, note that the functor $\mu_\sigma$ exactly covers all arrows labeled $\sigma$ in $\mu\mathrm{CPC}$. The extra structure imposed by the functor construction is the mapping between the arrows of $\kappa\mathrm{CPC}$—in other words, the functoriality of the multipliers expresses the compatibility of substitution with the reduction relation.

Corollary 8.3. The above specification of the multipliers satisfies all of $\mu\cdot1$, $\mu\cdot2$ and $\mu\cdot4$.

Proof. Checking that $\mu\cdot1$ and $\mu\cdot2$ hold is straightforward from looking at the tree structure for clauses. The previous proposition guarantees $\mu\cdot4$. $\square$

8.3. Embedding at Contexts and Canonizing

Intuitively, the embedders $\eta$ are here formed by set union.

To capture this in the tree representation, we added, as in the case of Gröbner bases, a void place $((n+1))$, occupied by the (meaningless) constant $Y$. In the tree representation of clauses, concatenating a clause to another (or rather, replacing $Y$ with the clause to be embedded) may lead to multiple occurrences of the same literal. (Instead of viewing clauses as sets of literals, in CPC they are viewed as multisets.)

Therefore, CPC will be the same category as $\kappa\mathrm{CPC}$, with the exception that in addition it contains objects representing clauses with multiple occurrences of the same literal.

The additional arrows between the objects are simply the $e_{CD}$ between objects $C$ and $D$ with the same canonical form, and all compositions ensuing from the additional $e_{CD}$.

The functor $\kappa$ will be the mapping from that category CPC to $\kappa\mathrm{CPC}$ such that all multiple occurrences of any such literal are reduced to a single one. Moreover, if in any clause $C$ a literal $L$ occurs equipped with all possible truth values (classically just $L$ and $\neg L$; in the many-valued case all $L^w_i$ for $w_i \in W$), then all these literals $L^w_i$ are collapsed into $\top$. Finally, $\kappa$ maps any set of clauses containing $\top$ to simply $\{\top\}$.

That this mapping constitutes a functor follows immediately from the irrelevance of multiple occurrences for the (logical) derivation relation modeled by the reduction, the fact that any disjunction $\bigvee_{w \in W} L^w$ is equivalent to $\top$, and the fact that any disjunction containing $\top$ is equivalent to $\top$ itself.

The difference between this case and that of Gröbner basis construction is that here, the elements of $\kappa\eta\mathrm{CPC}$ are easily seen to be endofunctors on $\kappa\mathrm{CPC}$, as in the case of completion of term rewriting systems.

Corollary 8.4. The above specification of the embedders and canonization satisfies all of $\eta\cdot1$, $\eta\cdot2$, $\eta\cdot3$, $\kappa\cdot3$, $\kappa\cdot4$, $\kappa\cdot5$, $\kappa\cdot6$ and $\kappa\cdot7$. 
**Proof.** \( \kappa \)-3 follows from the fact that the logical consequence relation is compatible with the canonization defined above (compare the subsumption and tautology rules). All of \( \kappa \)-4, \( \kappa \)-5, \( \kappa \)-6 and \( \kappa \)-7 follow again from the construction of \( \text{CPC} \). \( \eta \)-1 and \( \eta \)-2 follow from the way the embedders were defined on the tree representation of clauses above. Finally, \( \eta \)-3 follows from the compatibility of the ordering (logical consequence) with the equivalence introduced. \( \square \)

8.4. NORMALIZING PATTERNS AND FINDING CRITICAL PAIRS

Because of our definition of the ordering \( >_\kappa \) (cf. Section 8.1), normalizing arrows in \( \kappa \text{CPC} \) always results in arrows with only one clause on the left-hand side, as these are the ones with minimal left-hand sides among all those that reflect logical consequence relations derived from given patterns.

Now, finding coproducts can be reduced to the case of having an embedder and a multiplier on one side and the trivial rule \( \{ \top \} \to \{ \top \} \) on the other.

This has not so much to do with the coproduct structure in this case as the nature (or strategy) of the algorithm: instead of looking for a completed system, rather, we are looking for an arrow \( \{ \top \} \to \{ \bot \} \) and we therefore consider only coproducts of the object \( \{ \top \} \) with any object to the left-hand side of any pattern. As those objects will always correspond to sets with a single clause, this coproduct is \( \{ \top \} \) itself.

Note that the existence of the arrow \( \{ \top \} \to \{ \bot \} \) implies that every object has a normalizer \( \{ \bot \} \) (or \( \{ \top \} \); indeed, all objects are then isomorphic).

**Corollary 8.5.** *In the case of resolution all of \( \kappa \eta \)-1, \( \kappa \eta \)-2 and \( \kappa \eta \mu \) hold.*

**Proof.** \( \kappa \eta \)-1 follows from the fact that embedding (set union) respects the ordering and can be composed; \( \kappa \eta \)-2 is an immediate consequence of the fact that any set consisting of a single clause will always be considered minimal with respect to any of its logical consequences, and \( \kappa \eta \mu \) follows again from the composability (also among each other) of the multipliers and embedders. \( \square \)

8.5. EXAMPLE: RESOLUTION

**Example 8.1.** (Three-Valued Logic) Let us consider an example from Baaz (1992) for a three-valued logic with \( W = \{ t, u, f \} \) (for true, undecided, and false, respectively).

The original set of clauses \( \mathcal{C} \) is given as \( \{ \{ P^t(c), P^f(f(f(x))) \}, \{ P^u(x), P^t(f(x)) \}, \{ P^u(f(f(c))) \} \} \).

Let \( \kappa \text{CPC} \) be the corresponding reduction category. The first clause of \( \mathcal{C} \) corresponds to the basic arrows: \( r_{11} = \{ P^t(c) \to P^f(f(f(x))) \}, r_{12} = \{ P^u(c) \to P^f(f(f(x))) \}, r_{21} = \{ P^t(f(f(x))) \to P^t(c) \} \) and \( r_{22} = \{ P^u(f(f(x))) \to P^t(c) \} \).

The second clause generates: \( r_{31} = \{ P^f(f(x)) \to P^t(f(x)) \}, r_{32} = \{ P^t(x) \to P^t(f(x)) \}, r_{41} = \{ P^f(f(x)) \to P^u(x) \} \) and \( r_{42} = \{ P^u(f(x)) \to P^u(x) \} \).

Finally, the third clause generates: \( r_{51} = \{ P^f(f(f(c))) \to \{ \bot \} \} \) and \( r_{52} = \{ P^f(f(f(c))) \to \{ \bot \} \} \).

Moreover, we have the identity arrow \( r_0: \{ \top \} \to \{ \top \} \), which is also a basic arrow.

By embedding \( r_{51} \) with \( \{ P^f(f(f(c))) \}, P^u(f(f(c))) \} \) we obtain the following critical normalizer of \( r_{51} \) and \( r_0 \):
\[
\{\top\} = \{P^f(f(f(c))), P^u(f(f(c))), P^u(f(f(c)))\}
\]

\[
\{\top\} \xrightarrow{r_0} \{P^f(f(f(c))), P^u(f(f(c)))\}
\]

\[
\{P^u(f(f(c)))\}
\]

\[
\{P^u(f(c))\}
\]

As \{\top\} is the maximal element with respect to \(\succ\), we obtain \{\top\} \rightarrow \{P^u(f(c))\}, which gives rise to the rules: \(r_{61} = \{P^u(c)\} \rightarrow \{\bot\}\) and \(r_{62} = \{P^f(c)\} \rightarrow \{\bot\}\).

Now, we obtain the following critical normalizer of \(r_0\) and \(r_{21}\):

\[
\{\top\} = \{P^f(f(f(c))), P^u(f(f(c))), P^f(f(f(c)))\}
\]

\[
\{\top\} \xrightarrow{r_0} \{P^u(f(f(c))), P^f(f(f(c)))\}
\]

\[
\{P^u(f(c)), P^f(f(f(c)))\}
\]

\[
\{P^u(f(c))\}
\]

\[
\{P^u(f(c))\}
\]

In other words, every element \(C\) has the normal form \{\bot\}. (As \(C \rightarrow \{\top\} \rightarrow \{\bot\}\).) This shows that the system is unsatisfiable. Recall that the goal of the procedure is simply to construct \{\bot\} as a normal form for \{\top\}; when this is done, the terminal and initial object have been shown to be isomorphic and it is clear that \(\kappa\)CPC models an unsatisfiable system of clauses.

As can be seen from the example, the actual resolution steps are often hidden in the application of the multipliers and embedders. Therefore, several resolution steps may occur within one iteration of the (categorical) CPC version of the algorithm, so that there are fewer iterations than in the standard formulation of the algorithm.
9. Relation to Other Categorical Models of Rewriting

The closest model to ours is the one developed by Stell (1992), which uses a generalization of 2-categories, so-called sesqui-categories, for modeling aspects of unification and rewriting.

Among the categories defined there, we focus here on $\text{Fin}$, which is the category with as objects finite subsets of a given set of variables $V$, and as morphisms triples $(X, \phi, Y)$, where $\phi$ is a substitution such that its domain is contained in $X$ and the union of its codomain and the difference between $X$ and its domain is contained in $Y$. (This last set, $\text{cod}(\phi) \cup (X \setminus \text{dom}(\phi))$, corresponds to those variables of $Y$ that occur in terms of the form $\phi(x)$, for $x \in X$.)

In this setting, a distinction is made between two families of functors. Using Stell’s notation, the first is indexed by objects of the hom-set $\text{Fin}(W, X)$, consisting of arrows $g \circ_r: \text{Fin}(X, Y) \to \text{Fin}(W, Y)$; and the second is indexed by objects of the hom-set $\text{Fin}(X, Y)$, consisting of arrows $\circ_l: \text{Fin}(W, X) \to \text{Fin}(W, Y)$.

These correspond (in the term rewriting case which Stell studied) to our embedder and multiplier functors, respectively, in the sense that collecting the functors of the first family over all possible objects $W, X, Y$, one obtains all multiplier functors, and collecting those of the second family, one obtains all embedder functors. The chief difference between Stell’s approach and the one presented here is that we incorporated a possibility of canonization. This is necessary in order to incorporate the Gröbner basis algorithm and resolution in the general CPC algorithm (cf. Stokkermans, 1995a).

Reichel (1990) identified the basic constructions for the critical-pair/completion algorithm within a formal framework of 2-categories with products; these turn out to be the construction of pullbacks in the category of 0-cells and 1-cells, the horizontal decomposition of 1-cells, and the decomposition of 2-cells with respect to the direct product or the pairing of 2-cells. Based on those, critical pairs are characterized as narrowing a 2-cell against a 1-cell. The first two of these constructions are concerned with the structure of the entities, and captured by our multiplier and embedder functors; that these operations are functorial corresponds to the required properties in Reichel (1990). The third construction is replaced, in our model, by the construction of normalizers in $\kappa\mu\text{CPC}$; finally, the critical pairs then turn out to be characterized as reversed pushout diagrams (in $\kappa\mu\text{CPC}$) for weak coproducts of certain objects (determined by the left-hand sides of patterns) in $\kappa\eta\mu\text{CPC}$ (see also the remarks on the work by Rydeheard and Stell (1987) below).

Links with other approaches are much looser, mainly due to the different objectives that lie behind these models. For instance, Huet (1986) and Jay (1991) deal mainly with confluent systems (as opposed to making systems confluent, as in our case). In Huet [1986], the confluence of the system under consideration is characterized categorically by the existence of all pushouts, in a similar way to our use of “reversed” pushouts; however, Huet (1986) then proceeds to exclude all such term rewriting systems (namely non-left-linear) where this condition is violated. Jay (1991) is an analysis of already confluent systems: his definition of a category being weakly essentially normalizing corresponds to the relevant category having all normalizers in our approach.

The set-up in Bonacina and Hsiang (1991) is of a more general nature than that involved in our problem; instead of dealing with arbitrary inference rules (there characterized as certain natural transformations) we are only concerned with resolution (in the context of theorem proving) or, more precisely, the construction of additional rela-
tions (“derivations”) by means of critical pairs, of which resolution is a slightly modified version. Moreover, the primary goal there is characterizing search plans by means of functors, while we do not consider search strategies (i.e. strategies for selecting “fruitful” pairs of patterns at each iteration) in our model (although that is undoubtedly an interesting possible direction for further work).

The characterization of most general unifiers as (weak) coequalizers by Rydeheard and Stell (1987), which was the starting point for Stell’s work briefly described above, is replaced, in our model, by characterizing the so-called “most general superposition situations” (i.e. critical overlaps between rules in a TRS, LCM of leading monomials in a Gröbner basis under construction, and most general unifiers in the treatment of resolution) as weak coproducts in the auxiliary category $\mathcal{CPC}$. The difference is caused by our reluctance (in which we follow Jay) to introduce 2- and higher dimensional categorical constructions whenever avoidable; the justification for this lies in the large overhead of checking additional diagrams on commutativity, especially where the relevance of these extra conditions is not always easy to grasp intuitively when looking at the underlying CPC algorithms.

Finally, we make the following observation about the relationship of our work with Marché (1994) (though not a categorical approach): interpreting normalized rewriting as defined there in our categorical model would seem to amount to defining rewrites $A \rightarrow_n B$ iff $\kappa(A) \rightarrow B$ (where $\rightarrow_n$ denotes the normalized rewrite relation). In other words, one would obtain a categorical interpretation of normalized rewriting by changing the definition of $\kappa_{\mathcal{CPC}}$ accordingly. Identifying this “category of normalized rewriting” by $\nu_{\mathcal{CPC}}$, we obtain $\nu_{\mathcal{CPC}}$ from $\kappa_{\mathcal{CPC}}$ by allowing as domains of the (normalized) rewrite arrows only the canonical forms and adding the canonizers $\kappa: A \rightarrow \kappa(A)$ as arrows in $\nu_{\mathcal{CPC}}$.

10. Conclusion

We have introduced a general CPC algorithm formulated in the language of category theory. We have demonstrated how the Knuth–Bendix procedure, including its adaptation to handle TRS modulo equivalence relations, fulfills the requirements of this general framework. Our categorical CPC algorithm also encompasses the Gröbner basis algorithm and resolution (both for classical and many-valued logics; for the latter, cf. Baaz and Fermüller (1995)). It can be seen as a very substantial extension of the “common ancestor” of the completion of TRS and the Gröbner basis algorithm by Kandri-Rody et al. (1989) (The role of their simplification relation $\Rightarrow$, cf. Section 2, is taken by the canonizer functor $\kappa$ in our approach.) Another unifying approach for those two algorithms (but not resolution) can be found in Bündgen (1991b). In his approach, however, the TRS simulating the Gröbner basis algorithm is infinite, because an infinite set of rewrite rules is required to present $\mathcal{Q}$.

The general categorical model is of great potential interest in that one can now attempt to transfer parts of the great body of semantical knowledge in the theory of resolution through the categorical model (e.g. formulating the concept of semantical trees within the model) and apply it in TRS and polynomial ideal theory. Vice versa, one could try to use the multitude of optimization techniques in the theory of completion of TRS and bases of polynomial ideals by the reverse transformation in automated theorem proving. Both would be quite typical applications of the generality and expressiveness of category theory, and work in this direction is continuing.
Acknowledgements

I would like to thank Bruno Buchberger and Jochen Pfalzgraf for their support and encouragement, and Matthias Baaz, Barry Jay, Viorica Sofronie and John Stell for helpful discussions. I would also like to thank the anonymous referees for valuable suggestions on presentation. The XY-pic package of Kristofer Rose was used for some diagrams. This research was partially supported by the Austrian Science Foundation within ESPRIT BRP 6471 “MEDLAR II”.

References


A Categorical Critical-pair Completion Algorithm


Originally Received 13 January 1997
Accepted 24 January 1999