Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 87–105
www.elsevier.com/locate/entcs

Diagram Chase in Relational System Development

Michael Ebert[1] Georg Struth[2]

Department of Computer Science,
University of the Federal Armed Forces Munich, Germany

Abstract

We propose diagrammatic techniques for visualizing relational reasoning in formal methods like B or Z; in particular for induction and coinduction. These are similar to those for functional diagrams in category theory and inspired by rewriting theory. Diagrams are endowed with a simple algebraic semantics that imposes a convenient balance between expressive and algorithmic power. This makes the approach particularly suitable for mechanization and automation. Its usefulness for visual reasoning is illustrated by various examples.

Keywords: diagrams, semi-commutation, formal semantics, relational system development, B, Z, induction, coinduction, iterator algebras.

1 Introduction

Diagram chase with functions is common practice in mathematical sciences. A function $f : A \to B$ is associated with an arrow from source A to target B labeled by f. Composition of functions, a partial operation with respect to sources and targets or domains and codomains of functions, is associated with juxtaposition of arrows. An equation $f = g \circ h$ between functions is associated with a commuting diagram, where all paths along arrows from a given source to a given target are equal. Equational and diagrammatic reasoning about functions are in one-to-one correspondence. It is well-known that category theory provides an abstract foundation of this style. It is successfully used in

[1] Email: ebert@informatik.unibw-muenchen.de
[2] Email: struth@informatik.unibw-muenchen.de

doi:10.1016/j.entcs.2004.08.049

many different areas of computer science, among them the specification and analysis of software systems.

For many software analysis tasks, however, the functional approach is too restrictive. First, specifications are often non-deterministic, thus allow one-to-many associations of elements. Determinism of programs is usually established in a refinement process. Non-determinism is even an essential feature of concurrent or reactive systems. Second, functions are not entirely satisfactory for modeling the semantics of predicate transformers or of analysis formalisms like dynamic or temporal logics or the mu-calculus. It is therefore desirable to generalize from functions to relations. In fact, relations form the backbone of formal methods like B [3] or Z [18] in software development. But how to visualize relational reasoning?

We build on lessons learned from rewriting theory (c.f. [4]) and allegory theory [11] for developing a simple and convenient algebraic semantics for *semi-commuting* diagrams. These are for inequational reasoning with relations what commuting diagrams are for equational reasoning with functions. Our underlying algebra is Kleene algebra [14]. We believe that it is very convenient for modeling the control flow in software systems. In particular, this semantics yields simple fixpoint-based rules and associated diagrams for an interesting fragment of inductive and coinductive reasoning. This approach nicely balances the expressive power of diagrams with algorithmic power, notably decision procedures. A particular benefit is that regular and omega regular identities and their associated diagrams can freely be used in the diagram chase. This simplicity is in contrast to previous approaches that required considerable machinery.

The strength of our approach, however, consists not solely in the visualization of algebraic axioms by a sound and complete diagrammatic calculus, but also in intuitive and powerful derived rules for transforming and (de)composing diagrams, while preserving some interesting properties. We introduce rules that capture the typical informal diagrammatic style of rewriting theory. Our considerations in the general setting of semi-commutation, however, increases their domain of applicability to relational software development with methods like B or Z. The idea of transferring diagrammatic techniques from rewriting to software engineering is probably novel. Several examples from rewriting theory and concurrency control support the usefulnes of this approach.

The general merits of relational diagram chase in rewriting are beyond any doubt: it is the standard way of reasoning in this area. Diagrams show the essence of proofs while hiding boring technical details. Proofs by diagrams are informally rigorous in contrast to the formally rigorous proofs from logic or algebra. Our examples show that the present approach unifies both kinds

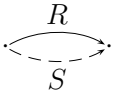
of rigor in one simple and convenient formalism. In a corresponding formal method, a proof engineer might first sketch a proof using diagrams and then incrementally refine it using our derived rules. The underlying algebraic semantics paves the way to a simple mechanized verification of the diagrammatic arguments. The approach is open in the sense that further diagram rules can easily be defined and verified by the user.

In this paper we try to informally motivate the main ideas behind our diagrammatic calculus and to illustrate them by examples. Theoretical results about Kleene algebras and related theories can be found in the literature [14,6,9]. An implementation of the calculus and an integration into a formal method is left for future work.

The remainder of this text is organized as follows: Section 2 shows how rewriting diagrams can be used for modeling meaningful properties of concurrent systems and processes. Section 3 proposes an algebraic semantics for semi-commuting relational diagrams. Section 4 presents transformation and preservation laws for diagrams. Sections 5 and 6 introduce diagrammatic induction techniques and examples. Section 7 gives diagrams and techniques for coinduction. Section 8 and 9 further discuss the results, draw a conclusion and suggests some future work.

2 Diagrams for Relation-Style Reasoning

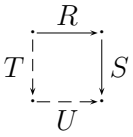
Let us first recall how diagrams arise in rewriting theory. Let R be a binary relation over some set A. We write $a \rightarrow_R b$ instead of $(a, b) \in R$ and visualize $\cdot \rightarrow_R \cdot$ as a basic diagrammatic building block for the entire relation R. Composition of relations R and S is visualized by $\cdot \rightarrow_R \cdot \rightarrow_S \cdot$. Remember that $a \rightarrow_{R \circ S} b$ if $a \rightarrow_R c$ and $c \rightarrow_S b$ for some $c \in A$. Finally, inclusion $R \subseteq S$ of R in S is visualized by the diagram



If $R \subseteq S$ is valid, then we say that the diagram *semi-commutes*.

The following example demonstrates the universality and applicability of this concept.

Example 2.1 Consider the following semi-commuting diagram.



It specializes to various interesting properties, among them the following.

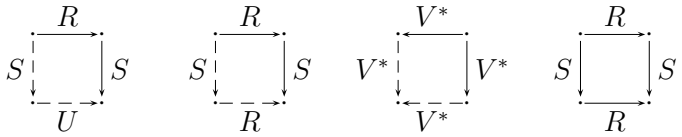
(i) Let $T = S$. Then we say that U *S-simulates* R. Simulation is a standard notion in concurrency theory and process algebra. By elementary set theory this simulation property holds in the following case: For all $x_1, x_2, x_3 \in A$, if $x_1 \rightarrow_R x_2$ and $x_2 \rightarrow_S x_3$ hold, then there is some $y \in A$ such that $x_1 \rightarrow_T y$ and $y \rightarrow_U x_3$. Therefore, in a semi-commuting diagram, paths of solid lines correspond to universal quantification and paths of dashed lines to existential quantification. This notation is also standard in rewriting theory. We will present several applications. As a special case, when T is a mapping, then the diagram says that T is a homomorphism.

(ii) Let moreover $U = R$. Then we say that S *semi-commutes* over R. In a concurrent system, this expresses that execution of S may always be given priority over execution of R; an R-sequence of actions followed by action S can always be replaced by an S-sequence of actions followed by action R. We will use several weaker notions of semi-commutation below.

(iii) Let $R^\circ = S = T = U^\circ = V^*$, where the operations * and $^\circ$ denote reflexive transitive closure and conversion. We visualize converses by inverting arrows. Then the diagram expresses *confluence* of V, that is all diverging V-paths will eventually join. This is a fundamental notion of rewriting and functional programming. It is used to ensure the existence of unique solutions of computing processes.

(iv) Let $\subseteq$ be equality; let $T = S$ and $U = R$. Then we say that the diagram *commutes* and we can replace the dashed lines by solid ones. *Commutation* of two relations R and S, that is, $R \circ S = S \circ R$ may be used to express their independent execution. This is interesting for analyzing concurrent systems, for instance for partial order reduction [1], for expressing independence of measurements in a physical system or for modeling independence of variable assignments in a program.

The diagrams that correspond to these notions are



3 Abstract Semantics for Semi-Commuting Diagrams

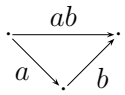
We now abstract from set-theoretic relations to arrows labeled with elements of some suitable algebra and motivate the axioms of our labeling algebra step by step. We model diagrams as special digraphs.

Consider any directed acyclic graph G with one source and one sink, with two sorts of edges (or arrows)—solid and dashed—and with arrows labeled by the elements of some labeling algebra A. A *link* is a path from source to sink that contains only one sort of arrows. Then G is a *diagram* if all paths from source to sink are links.

We define commutation of a diagram D via path equivalence. So we require an operation of multiplication of labels (denoted by juxtaposition). For every path π in D we define $l(\pi)$, the label associated with path π, as the homomorphic extension of the mapping that associates arrows with their labels and the empty path with 1 and such that $l(\pi\rho) = l(\pi)l(\rho)$. Here, juxtaposition of paths denotes path composition. Obviously, l is a bijection if multiplication of labels is associative and l maps to the respective equivalence classes. We will impose this requirement. For all π_1 and π_2 in D, the relation $\pi_1 \sim_A \pi_2$ defined by $l(\pi_1) = l(\pi_2)$ is a congruence (with respect to path composition), which we call *path congruence*. We say that D *commutes* if all links are (path-)equivalent. We write $C(\pi_1, \ldots, \pi_n)$ if the diagram with links $\pi_1, \ldots, \pi_n$ commutes. Thus

$$C(\pi_1, \ldots, \pi_n) \Leftrightarrow \pi_i \sim_A \pi_j, \quad \text{for all } 1 \leq i, j \leq n.$$

Example 3.1 The following diagram commutes.



In our intended applications, arrows correspond to actions; a meaningful notion of path in a diagram presupposes that composition of actions is associative. Composition of relations, of course, is.

It is also convenient to have arrows that preserve or destroy everything. They correspond to the actions skip and abort and, alternatively, to the identity and the empty relation. These actions occur in many formalisms, among them predicate transformer calculi, propositional dynamic logic and process algebras. We introduce labels 1 and 0 and require commutation of

This turns the labeling algebra into a monoid with unit 1 and element 0 that satisfies the annihilation laws $0a = 0 = a0$. Some applications suggest to drop $a0 = 0$. When a is always non-terminating, one might want that $ab = a$ for all b and in particular for $b = 0$; a conflict. See [21] for a deeper discussion.

It is convenient to model a concurrent system as a finite or infinite iteration of a choice of actions. We therefore add an operation $+$ of addition to our labeling algebra that is associative, commutative and idempotent ($x + x = x$). At the relational level, addition is set-union. Abstracting from relations we require that 0 is a unit of addition and that multiplication distributes over addition from left and right. Operations for iteration are considered below.

To sum up, we require that the labeling algebra $(A, +, \cdot, 0, 1)$ is an idempotent semiring. Formally, this means that

- $(A, +, 0)$ is a commutative idempotent monoid, that is, a semilattice,
- $(A, \cdot, 1)$ is a monoid,
- multiplication distributes over addition from left and right
- zero is a left and right annihilator.

The definition of semi-commuting diagrams is now straightforward. Every idempotent semiring A possesses a natural ordering defined by $a \leq b$ iff $a + b = b$ for all $a, b \in A$. It is the only ordering with least element 0 and for which addition and multiplication are isotone. Now for paths π_1 and π_2 in a diagram D we define $\pi_1 \preceq \pi_2$ iff $l(\pi_1) \leq l(\pi_2)$. By isotonicity of multiplication, $\preceq$ is a path pre-congruence. We say that D *semi-commutes* if every solid link in D is smaller than every dashed link in D with respect to $\preceq$. Of course, semi-commutation of a diagram can be defined in terms of commutation and vice versa. We write $S(\pi_1, \ldots, \pi_m; \rho_1, \ldots, \rho_n)$ to denote that a diagram with solid links $\pi_1, \ldots, \pi_m$ and dashed links $\rho_1, \ldots, \rho_n$ semi-commutes. Therefore,

$$S(\pi_1, \ldots, \pi_m; \rho_1, \ldots, \rho_n) \Leftrightarrow \pi_i \preceq \rho_j, \quad \text{for all } 1 \leq i \leq m \text{ and } 1 \leq j \leq n.$$

By the natural ordering, idempotent semirings allow reasoning about program and system refinement. $a \leq b$ holds iff a is a refinement of b. This is the case since a can always be used instead of b in all contexts.

Since l is a homomorphisms of diagrams into the labeling algebra and semi-commutation is defined via this interpretation in terms of validity of inequalities, the labeling algebra provides a semantics for semi-commuting diagrams. Since l is in fact bijective (up to associativity of multiplication),

semi-commuting diagrams are a visualization of the algebra. Moreover, by this simple construction, it is obvious that the semi-commuting diagrams that can be built from the axioms of the idempotent semirings are sound and complete with respect to this semantics.

4 Transformation and Preservation

The simple one-to-one correspondence between diagrams and algebraic inequalities allows us to express theorems of the labeling algebra immediately in terms of semi-commuting diagrams. In particular, valid quasi-identities in the theory of idempotent semirings, that is formulas of the form

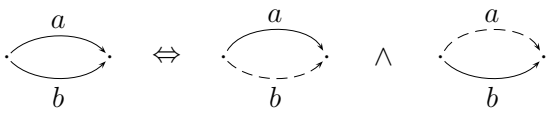
$$a_1 \leq b_1 \wedge \cdots \wedge a_n \leq b_n \Rightarrow a_0 \leq b_0,$$

translate immediately into preservation laws for semi-commutation:

$$S(l^{-1}(a_1); l^{-1}(b_1)) \wedge \cdots \wedge S(l^{-1}(a_n); l^{-1}(b_n)) \Rightarrow S(l^{-1}(a_0); l^{-1}(b_0))$$

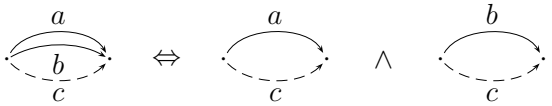
and vice versa. This law says that whenever the diagrams in the antecedent semi-commute, then so does the succedent. Any proof system for equational logic can therefore serve as a proof system for diagrams. Experience, however, shows that human reasoning with idempotent semirings is essentially inequational. It is therefore more convenient to use a logic for reasoning with orderings and precongruences. See [22] for such systems. Consequently, reasoning about semi-commuting diagrams can easily be mechanized and integrated into a formal method. Since the equational theory of idempotent semirings is decidable, it is decidable whether a given diagram semi-commutes. However, implications between diagrams can be undecidable; a consequence of undecidability of the uniform word problem for semigroups [7]. So large parts of diagrammatic reasoning can easily be automated; the remainder may require human interaction. Thus the algorithmic power of the labeling algebra is very helpful for checking the validity of diagrammatic reasoning.

Nevertheless, a plain general-purpose formal proof system is inadequate for diagram chase. Truly diagrammatic reasoning, as category theory shows, should be based on derived transformation rules that are topologically intuitive and natural and that preserve the relevant properties, such as commutation or semi-commutation. We will present two kinds of such rules: rules for replacing paths in diagrams and rules for composing and decomposing diagrams. We first present splitting and denesting rules. The splitting rule allows us to translate between commuting and semi-commuting diagrams.



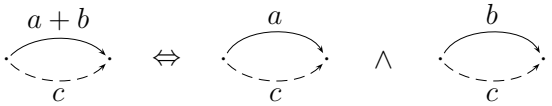
Obviously, it is valid, since $a = b$ iff $a \leq b$ and $b \leq a$.

The denesting rule yields a more compact representation of diagrams, when used from left to right. In the converse direction, it introduces further normal-form diagrams that consist of precisely two links.



A similar rule can be obtained by exchanging solid and dashed lines.

Our second derived rule is a case analysis rule



It is valid, since $a + b \leq c$ iff $a \leq c$ and $b \leq c$ holds for all elements a, b, c of some idempotent semiring. Using distributivity, we can further decompose all diagrams into a normal form where all solid arrows are labeled by single letters. There is no corresponding law for dashed lines. Because of this case analysis, we do not introduce any diagrammatic laws for addition in the context of dashed lines, but handle addition entirely by properties of the labeling algebra. One can easily derive further diagrammatic proof rules that translate statements such as $a \leq b \Rightarrow a \leq b + c$ or $a \leq a + b$.

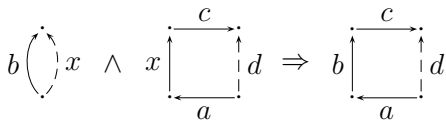
Our third set of derived rules transforms paths in diagrams. They are based on isotonicity of multiplication, that is the derived rules

$$\pi_1 \preceq \pi_2 \Rightarrow \rho\pi_1 \preceq \rho\pi_2,$$
$$\pi_1 \preceq \pi_2 \Rightarrow \pi_1\rho \preceq \pi_2\rho.$$

on path expressions π_1, π_2 and ρ and the associated preservation laws for semi-commutation

$$S(\pi_1; \pi_2) \Rightarrow S(\rho\pi_1; \rho\pi_2),$$
$$S(\pi_1; \pi_2) \Rightarrow S(\pi_1\rho; \pi_2\rho).$$

Although these isotonicity rules are algebraically fundamental, they are not natural for a diagrammatic calculus. We now give two fundamental rules for replacing edges in semi-commuting diagrams. The first one is



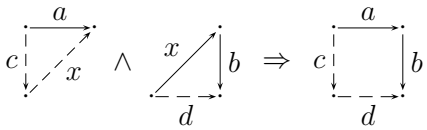
The second rule can be obtained by exchanging solid and dashed lines. Using these rules and the splitting rule, we can derive another rule for commuting diagrams that has the same shape and in which all lines are solid. There are further derived diagram rules for combinations of commuting and semi-commuting diagrams that can be derived from the fundamental rules using the splitting rule. For the sake of brevity we present them using path expressions.

$$\pi_1\rho_1\pi_2 \sim \pi_3 \wedge \rho_2 \preceq \rho_1 \Rightarrow \pi_1\rho_2\pi_2 \preceq \pi_3,$$
$$\pi_1 \sim \pi_2\rho_1\pi_3 \wedge \rho_2 \preceq \rho_1 \Rightarrow \pi_1 \preceq \pi_2\rho_2\pi_3,$$
$$\pi_1\rho_1\pi_2 \preceq \pi_3 \wedge \rho_1 \sim \rho_2 \Rightarrow \pi_1\rho_2\pi_2 \preceq \pi_3,$$
$$\pi_1 \preceq \pi_2\rho_1\pi_3 \wedge \rho_1 \sim \rho_2 \Rightarrow \pi_1 \preceq \pi_2\rho_2\pi_3.$$

Their translation into preservation laws is also straightforward. The second transformation rule, for instance, translates into the preservation law

$$C(\pi_1, \pi_2\rho_1\pi_3) \wedge S(\rho_2; \rho_1) \Rightarrow S(\pi_1; \pi_2\rho_2\pi_3).$$

Our fourth set of derived diagram rules deals with the composition of diagrams. Again, these rules are based on isotonicity of multiplication, but now contexts are used in both antecedents. Again, there are two fundamental composition rules for semi-commuting diagrams. The first one is



The second rule can again be obtained by replacing solid and dashed lines. A corresponding rule for commuting diagrams follows immediately from those for semi-commutation and the splitting rule. Further derived rules handle again the combination of commutation and semi-commutation. They are

$$\pi_1 \sim \pi_3\rho \wedge \rho\pi_2 \preceq \pi_4 \Rightarrow \pi_1\pi_2 \preceq \pi_3\pi_4,$$
$$\pi_1 \preceq \pi_3\rho \wedge \rho\pi_2 \sim \pi_4 \Rightarrow \pi_1\pi_2 \preceq \pi_3\pi_4,$$
$$\pi_1\rho \sim \pi_3 \wedge \pi_2 \preceq \rho\pi_4 \Rightarrow \pi_1\pi_2 \preceq \pi_3\pi_4,$$
$$\pi_1\rho \preceq \pi_3 \wedge \pi_2 \sim \rho\pi_4 \Rightarrow \pi_1\pi_2 \preceq \pi_3\pi_4.$$

Soundness of all these derived diagram rules can easily be shown with the help

of the labeling algebra. Moreover, all rules correspond to preservation laws for commutation and semi-commutation.

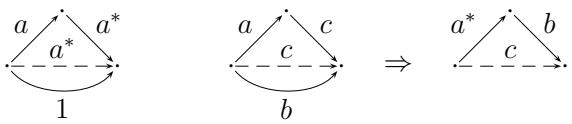
Our derived transformation and composition rules formalize the essence of diagram chase in rewriting theory, but generalized to the setting of semi-commutation and with an underlying abstract algebraic semantics. Diagram chase is usually far more convenient for a human than performing the detailed underlying algebraic manipulations (except for sectarians of idempotent semirings). The simple formal semantics of the present approach yields a coincidence of these opposites.

5 Diagrammatic Techniques for Induction

Reasoning by diagrams is often inductive. Reasoning about programs and systems usually requires some form of iteration or recursion. We will now consider a restricted form of induction that arises from finite iteration. Although strictly less expressive than induction corresponding to general recursion, it still captures some interesting behavior. A previous approach to inductive reasoning with geometric diagrams has been proposed in [12]. However, the goals and techniques used in this approach are entirely different from ours. To model induction in our labeling algebra, we extend the labeling algebra from idempotent semirings to a Kleene algebra [14]. Formally, this is a structure $(K, ^*)$ such that K is an idempotent semiring and the *star* * is a unary operation defined by the *star unfold law* and the *star induction laws*

$$1 + aa^* \leq a^*, \qquad b + ac \leq c \Rightarrow a^*b \leq c, \qquad b + ca \leq c \Rightarrow ba^* \leq c,$$

for all $a, b, c \in K$. The star unfold law and the first star induction law correspond to the diagrammatic expressions

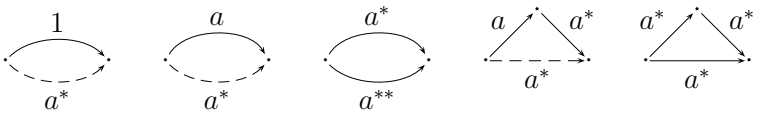


The diagram for the second star induction law is obtained by inverting arrows. Alternatively, also the induction laws $ac \leq c \Rightarrow a^* \leq c$ and $ca \leq c \Rightarrow a^* \leq c$ and the associated diagrams can be used.

Let us briefly discuss these laws. The unfold law says that an iteration of a either does nothing or performs an a-step and then continues the iteration. The induction law implies (by setting $b = 1$) that a^* is the least element with this property. It follows that a^* is the reflexive transitive closure of a, reflexivity meaning that $1 \leq a$ and transitivity that $aa \leq a$. Also a transitive

closure operator $^+$ can easily be defined as $a^+ = aa^*$, whence $a^* = 1 + a^+$. Operationally, the star induction law is a star elimination law. We will see examples below.

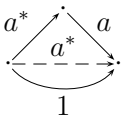
The extension to Kleene algebra preserves the nice algorithmic properties of diagram chase. The equational theory of Kleene algebra is still decidable using finite automata and a correspondence between Kleene algebra expressions and regular expressions [14]. One can therefore freely use the well-known *regular identities* and the associated *regular diagrams*. Examples are



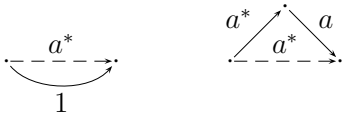
Finally, the reflexive transitive closure of a relation satisfies the star unfold and induction laws. More generally, the set-theoretic relations under union, relational composition and reflexive transitive closure form a Kleene algebra, which we call the *relational model*.

6 Examples for Induction

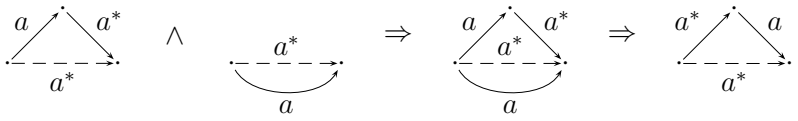
We now present several examples for inductive reasoning with diagrams. First, we show a dual star unfold law that follows from the first one by inverting arrows:



The proof is the following diagram chase. As common in theorem proving, we move backward from the conclusion towards the assumptions. First, we use the denesting rule. This yields the diagrams

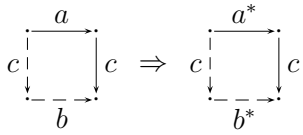


The first diagram follows immediately from the denesting rule and the star unfold law of Kleene algebra. For the second diagram, we reason as follows.

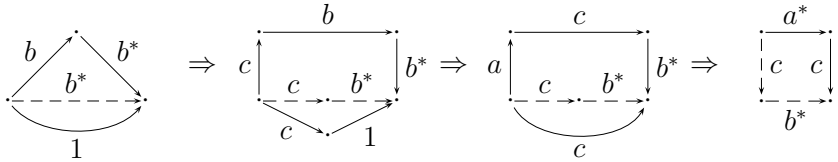


The last step in the chain of implications (and the first step in our backwards proof) uses the star induction law as a star elimination rule. The remaining step is by denesting and yields two new goals. The first goal follows immediately from the star unfold diagram and denesting. The second goal is one of our previous regular diagrams, which we do not analyze any further. The underlying proof in Kleene algebra can easily be recovered. The entire goal could as well be handled effectively by the decision procedure for regular expressions.

Our second example considers a simulation property which, as a derived rule, is useful in many applications in program transformation, refinement and concurrency theory:

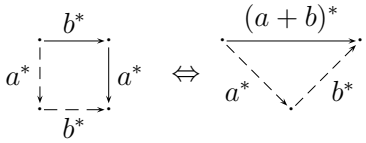


Its proof is as follows.

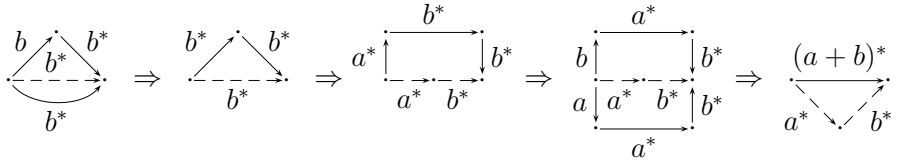


The first diagram visualizes the star unfold law. The first step uses isotonicity. The second step uses the assumption and unit elimination. The third step uses star induction. Our notion of *simulation* is consistent with that from process theory. For Kleene algebra K and $a, b, s \in K$ we say that b s-*simulates* a if $as \leq sb$. There is an interesting connection with data refinement, as considered in [21].

Our third example is a *separation theorem*. It states conditions for taking actions of a concurrent system $(a+b)^*$ as atomic units. Consider the property



This is a variant of the Church-Rosser theorem from term rewriting. See [19] for an extensive algebraic treatment of this example. The most interesting part of the proof is to show that $ba^* \leq a^*b^* \Rightarrow (a+b)^* \leq a^*b^*$:



The first diagram uses the regular identity $bb^* \leq b^*$ and reflexivity. The first step uses star induction. The second step uses isotonicity. The third step uses the assumption and star unfold for the new diagram. The fourth step uses star induction and case analysis. Again the proof should be read backwards. Setting b^* to the converse of a^* recovers the standard textbook proof of the Church-Rosser theorem from rewriting theory. The diagram corresponding to $ba^*b^* + aa^*b^* \leq a^*b^*$ yields the induction step.

It has been shown in [19] that more theorems of abstract rewriting can be proved in Kleene algebra, among them the abstract parts of the two standard proofs of the Church-Rosser theorem in the lambda calculus. This includes Barendregt's strip lemma, the Hindley-Rosen commutation lemma and the Hindley-Rosen commutative union lemma. In all cases, the standard diagrammatic arguments can immediately be translated into algebra and vice versa and verified with respect to our simple semantics. Previously, Nipkow has formalized a notion of diagram in the ISABELLE proof-checker that is based on set-theory [17]. His approach uses much more technical machinery.

7 Coinduction: Diagrams and Techniques

Kleene algebra can be extended by an operator for (strictly) infinite iteration. An *ω-algebra* [6] is a structure (K, ω) such that K is a Kleene algebra and $^\omega$ satisfies the *omega unfold law* and the *omega coinduction law*

$$a^\omega \leq aa^\omega,$$
$$c \leq b + ac \Rightarrow c \leq a^\omega + a^*b,$$

for all $a, b, c \in K$. The translation into diagrams is obvious. Again, the relational model of Kleene algebra with infinite iteration is an ω-algebra and the ω-operation is isotone with respect to the natural ordering. In contrast to Kleene algebra, ω-algebra allows one to express termination of iteration: It can be encoded as absence of infinite iteration, that is $a^\omega = 0$. In [6] it is claimed that the valid identities of ω-algebra are precisely the valid identities between ω-regular expressions. Therefore, we freely use the the well-known

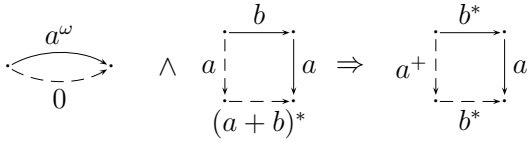
ω-regular identities and the associated diagrams in proofs. We will refer to them whenever they appear.

We now present two examples for coinductive diagrammatic reasoning. In the previous section, we have presented a separation theorem that was based on a semi-commutation condition. Without this condition we at least obtain the following separation property [20]:

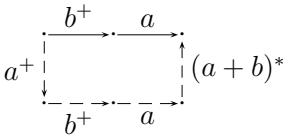
$$(a+b)^* = a^*b^* + a^*b^+a(a+b)^*.$$

It splits an arbitrary sequence of actions a and b into a "good" part where all a-actions are executed prior to b-actions and into a "bad" part that contains reversals of the form ba. Our first example compares two different conditions for separating actions. The first one is the *semi-commutation* condition $b^*a \leq a^+b^*$; the second one is the *quasi-commutation* condition $ba \leq a(a+b)^*$. With the simulation law from our earlier example it is easy to show that quasi-commutation is equivalent to $b^*a \leq a(a+b)^*$. It is also easy to show that semi-commutation implies quasi-commutation. For the converse implication, the termination condition $a^\omega \leq 0$ is needed.

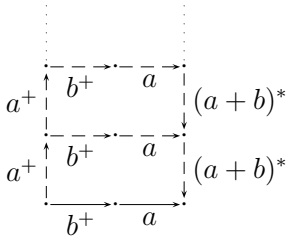
Thus our first example is to show that quasi-commutation and termination of a implies semi-commutation. Diagrammatically, this is expressed as follows:



We now present a semi-formal diagrammatic argument that can later be translated into a formal proof in ω-algebra. So let us assume that the antecedent holds, but the succedent does not, whence in particular $b^+a \not\leq 0$. It then follows from the above separation into good and bad sequences together with set theory that

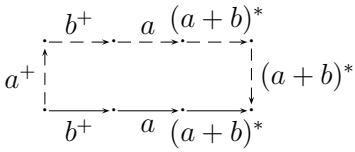


Iterating this argument yields a Jacob's ladder with an infinite a-chain (the previous diagram turned upside down in order to unfold towards heaven).

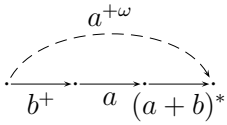


Note that this Jacob's ladder is not a semi-commuting diagram in the formal sense. The dotted lines indicate that it rather represents a family of semi-commuting diagrams that are constructed stepwise by the iteration.

Formally, the construction of this ladder corresponds, up to isotonicity, to the application of the omega unfold law to the diagram



This infinite iteration is captured formally by applying the omega coinduction law. It yields the semi-commuting diagram



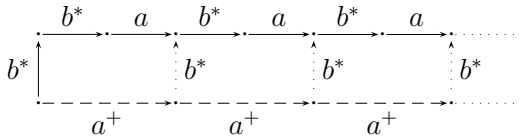
Now, by the omega regular diagram for $a^{+\omega} = a^\omega$, the assumption $a^\omega \leq 0$ and our fundamental rule for replacing edges in semi-commuting diagrams, we can replace the dashed line by another dashed line that is labeled with 0. By similar arguments, using the regular diagram for $1 \leq a^*$ and our unit diagrams, we can replace the solid link by another solid link that is labeled by b^+a. The resulting diagram expresses a contradiction. It therefore follows that $b^*a \leq a^+b^*$.

This intuitive argument requires set-theoretic reasoning beyond omega algebra, since the separate consideration of good and bad sequences works in set theory, but not for general least upper bounds. A very similar proof within omega algebra has been given in [20]. There, a context denoting the good sequences is carried along the proof. This style of reasoning, however, is less appropriate for diagrams, since a sum that appears at the right-hand side of the omega coinduction rule does not allow a modular treatment. This further underlines the usefulness of integrating our diagrammatic approach into

a formal method for set-based program development.

Our second example shows that termination of a implies termination of b^*a if a quasi-commutes over b. In this example we show again how to recover an algebraic proof from a semi-formal diagram chase. Like in the previous example, we use dotted lines and arrows for representing the iterative construction of a family of growing diagrams.

For the proof, assume an infinite b^*a-chain by reductio ad absurdum. Then, by the previous example, every b^*a-sequence can be transformed into an a^+b^*-chain if a terminates and a quasi-commutes over b. Thus the diagram



yields an infinite a-sequence, a contradiction. In order to recover a semi-commuting diagram, the inner dotted arrows should be discarded and the final dotted arrow should be replaced by a dashed one. Thus, in a sense, the dotted arrows memorize the previous steps of the iterative construction of semi-commuting diagrams.

The following algebraic proof is a direct translation that verifies the diagrammatic proof. First, by our previous example, we may assume semi-commutation instead of quasi-commutation, since a terminates. Therefore

$$(b^*a)^\omega = b^*a(b^*a)^\omega \leq a^+b^*(b^*a)^\omega = a^+(b^*a)^\omega.$$

The first step uses omega unfold. The second step uses the assumption. The third step uses the omega-regular identity $b^*(b^*a)^\omega = (b^*a)^\omega$. Now instead of iterating this construction by omega unfold (which we did not explicitly show in the diagram), we make the infinite chain explicit by omega coinduction, which yields $(b^*a)^\omega \leq a^{+\omega}$. Then the claim $(b^*a)^\omega \leq 0$ follows by the omega regular identity $a^{+\omega} = a^\omega$ and the termination assumption $a^\omega \leq 0$. We leave the full diagrammatic development to the reader. In [20], this theorem is used in an algebraic proof of the following modularity theorem for termination of action sequences from [5]: a and b terminate iff $a + b$ terminates and a quasi-commutes over b. This demonstrates the applicability of our diagrammatic techniques in termination analysis.

8 Discussion

Our considerations suggest that our algebraic approach to diagram chase is very suitable for integration into a formal method. It nicely balances expres-

sive and algorithmic power. Meanwhile, the underlying algebras have been implemented in various interactive theorem provers [19,21,13], a specialized interactive theorem prover for Kleene algebra [2] has been developed and the integration of decision procedures in such provers is in progress [8]. Proofs with such tools usually proceed backwards from the goal to the hypotheses. The star induction rule, for instance, is used as a star elimination rule. It is therefore straightforward to implement the semantics of our derived diagram transformation rules in an interactive theorem prover. Backward use of our transformation rules supports decompositional reasoning. Diagrams are iteratively narrowed: backward replacement of solid arrows yields greater edges; that of dashed arrows yields smaller edges with respect to the natural ordering. This corresponds to the usual style of inequational reasoning.

Therefore, at the present stage, the main task is the design and implementation of a graphical front-end that allows the explicit manipulation of diagrams, supports user-defined transformation rules and enhances a transition between algebraic and diagrammatic reasoning. A main question is proof presentation. At first sight, its seems most natural to modify diagrams successively on the screen by applying transformation rules in a drag an drop style. The concrete realization of such a tool is left open.

Our approach is motivated by relational reasoning. In set-based program development methods, this is usually the preferred level of abstraction. Therefore, an integration into methods like Z or B seems very interesting.

9 Conclusion

We have presented diagrams for relation-style reasoning with a formal semantics based on variants of Kleene algebra. We have demonstrated the usefulness of the approach by a series of examples from concurrency and rewriting theory that included inductive and coinductive reasoning. We have argued that the approach is particularly suitable for mechanization and automation. Large parts of reasoning use regular and omega regular identities that can be decided by automata.

We envision the following further work. First, our techniques should be integrated into a graphical interface that allows drag and drop diagram chase in combination with algebraic calculations. This interface should then be included into a formal method by implementing the semantics of diagrams. This novel integration of rewriting-style diagram techniques into software engineering methods like B or Z might contribute to increase their comfort. Second, other variants of Kleene algebras, for instance refinement algebras [21], lazy Kleene algebras [16] and typed Kleene algebras should be considered as a se-

mantics for reasoning diagrammatically about total correctness. Third, the approach should be generalized to Kozen's Kleene algebra with tests [15] and to modal Kleene algebras and algebras of modal operators [9,10]. This would support reasoning by diagrams about partial correctness and with formalisms such as dynamic or temporal logics. Fourth, it seems very interesting to consider generalized forms of diagrams, for instance with several sources and sinks and diagrams in which only parts commute or semicommute. Finally, the techniques developed in this text should be further tested in the analysis and development of algorithms, programs and protocols.

Acknowledgment: The authors would like to thank the anonymous referees and the participants of the VLFM'04 for valuable discussions.

References

[1] Abdulla, P. A., B. Jonsson, M. Kindahl and D. Peled, *A general approach to partial order reductions in symbolic verification*, in: A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV'98*, LNCS **1427** (1998), pp. 379–390.

[2] Aboul-Hosn, K. and D. Kozen, *KAT-ML: An interactive theorem prover for Kleene algebra with tests*, in: B. Konev and R. Schmidt, editors, *4th International Workshop on the Implementation of Logics*, Technical Report ULCS-03-018, University of Liverpool, 2003, pp. 2–12.

[3] Abrial, J.-R., "The B-Book," Cambridge University Press, 1996.

[4] Baader, F. and T. Nipkow, "Term Rewriting and All That," Cambrigde University Press, 1998.

[5] Bachmair, L. and N. Dershowitz, *Commutation, transformation, and termination*, in: J. H. Siekmann, editor, *8th International Conference on Automated Deduction*, LNCS **230** (1986), pp. 5–20.

[6] Cohen, E., *Separation and reduction*, in: R. Backhouse and J. N. Oliveira, editors, *Proc. of Mathematics of Program Construction, 5th International Conference, MPC 2000*, LNCS **1837** (2000), pp. 45–59.

[7] Davis, M., "Computability and Unsolvability," McGraw-Hill, 1958.

[8] Desharnais, J. (2004), personal communication.

[9] Desharnais, J., B. Möller and G. Struth, *Kleene algebra with domain*, ACM Transactions on Computational Logic (2004), to appear.

[10] Desharnais, J., B. Möller and G. Struth, *Termination of modal Kleene algebra*, in: J. Mitchell, editor, *IFIP-TCS 2004* (2004), to appear.

[11] Freyd, P. and A. Scedrov, "Categories, Allegories," North-Holland, 1990.

[12] Jamnik, M., "Mathematical Reasoning with Diagrams: From Intuition to Automation," CSLI Press, 2001.

[13] Kahl, W., *Calculation relation-algebraic proofs in Isabelle/Isar*, in: R. Berghammer, B. Möller and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, LNCS **3051** (2004), pp. 178–190.

[14] Kozen, D., *A completeness theorem for Kleene algebras and the algebra of regular events*, Information and Computation **110** (1994), pp. 366–390.

[15] Kozen, D., *Kleene algebra with tests*, Trans. Programming Languages and Systems **19** (1997), pp. 427–443.

[16] Möller, B., *Lazy Kleene algebra*, in: D. Kozen, editor, *Mathematics of Program Construction : 7th International Conference, MPC 2004*, LNCS (2004), to appear.

[17] Nipkow, T., *More Church-Rosser proofs (in Isabelle/HOL)*, J. Automated Reasoning **26** (2001), pp. 51–66.

[18] Spivey, J. M., "Understanding Z," Cambrigde University Press, 1988.

[19] Struth, G., *Calculating Church-Rosser proofs in Kleene algebra*, in: H. de Swart, editor, *Relational Methods in Computer Science, 6th International Conference*, LNCS **2561** (2002), pp. 276–290.

[20] Struth, G., *Abstract abstract rewriting*, Journal of Logic and Algebraic Programming (2004), to appear.

[21] von Wright, J., *From Kleene algebra to refinement algebra*, in: E. A. Boiten and B. Möller, editors, *Mathematics of Program Construction : 6th International Conference, MPC 2002*, LNCS **2386** (2002), pp. 233–263.

[22] Wechler, W., "Universal Algebra for Computer Scientists," Springer, 1992.