Combination of Constraint Systems

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Philosophie an der Ludwig-Maximilans-Universität
München

vorgelegt von

Stephan Kepser

München
1998

parentibus meis

Contents

Zusammenfassung

Den Inhalt der Dissertation bilden verschiedene Aspekte der Kombination von Constraint-Systemen. Der Begriff „Constraint“ läßt sich im Deutschen mit Neben- oder Seitenbedingung nur unvollständig wiedergeben. Daher lassen wir ihn als Terminus technicus hier unübersetzt. Ein Constraint-System in unserem Sinne besteht aus drei Teilen: Einer Constraint-Sprache, einem Lösungsbereich und einem Constraint-Löser. Die Constraint-Sprache ist die Sprache, in der die Probleme formuliert werden. Typischerweise ist sie eine Sprache erster Ordnung oder ein Fragment davon. Der Lösungsbereich ist eine algebraische Struktur, in der die Constraint-Sprache interpretiert wird. Wir betrachten dabei bestimmte Lösungsbereiche, nämlich sogenannte quasi-freie Strukturen, die weiter unten erläutert werden. Der Constraint-Löser schließlich entscheidet ein Fragment der Constraint-Sprache, das heißt, er entscheidet für jede Formel des Fragments, ob die Formel in der Lösungsstruktur wahr ist. Das betrachtete Fragment ist dabei meist das existentielle, das heißt, die Formeln sind Konjunktionen und Disjunktionen von Atomformeln oder negierten Atomformeln. Die in den Formeln auftretenden Variablen werden als implizit existenziell quantifiziert betrachtet. Die Formel ist demnach wahr in der Lösungsstruktur, wenn es eine Abbildung gibt, die den Variablen der Formel Elemente der Lösungsstruktur in der Weise zuordnet, daß die Formel in der Struktur gilt.

Betrachtet man ein Constraint-System derart abstrakt, so stellt sich die Frage, was für *systematische* Methoden zur Kombination von Constraint-Systemen es gibt. Wichtig ist uns dabei, daß die Methoden allgemeingültig und prinzipieller Natur sind; Spezialverfahren interessieren uns weniger. Eine systematische Methode hat demnach folgendes zu leisten: Einmal muß sie für zwei beliebige Lösungsbereiche ein Verfahren zur Konstruktion eines kombinierten Lösungsbereiches zur Verfügung stellen. Dabei soll der kombinierte Lösungsbereich möglichst allgemeingültig sein, und er sollte wesentliche strukturelle Eigenschaften der Komponentenbereiche mit diesen teilen. Zum zweiten muß die Methode einen Algorithmus beinhalten, der die Lösung von gemischten Constraints über der kombinierten Struktur erlaubt, also die beiden Constraint-Löser der Komponenten zusammenbindet und gemischte Constraints über der vereinigten Constraintsprache in reine Constraints je einer Sprache reduzieren kann, und zwar in solcher Weise, daß es für die reinen Constraints unter Benutzung der Constraint-Löser der jeweiligen Komponenten genau dann eine Lösung gibt, wenn es auch für die gemischten Constraints eine Lösung in der kombinierten Lösungsstruktur gibt. Wir verlangen weiterhin Konservativität in folgendem

Sinne: Die Lösung eines reinen Constraints aus einer Komponentensprache soll in der kombinierten Lösungsstruktur zu keinem neuen Resultat führen, ein solcher Constraint soll in der kombinierten Struktur genau dann gelten, wenn er in der Komponentenstruktur gilt.

Wenn man den Begriff von Kombination so allgemein und weit faßt, wie wir es tun, ist es erforderlich, gewisse Einschränkungen bei den Komponentenconstraint-Systemen zu machen. Die erste wichtige Einschränkung betrifft die Signaturen der Komponentenconstraint-Sprachen. Mit Signatur bezeichnet man die Menge der Konstanten, Funktionssymbole und Relationssymbole, aus der die Constraint-Sprache aufgebaut wird. Wir verlangen, daß die Signaturen der Komponentensprachen disjunkt sind, das bedeutet, daß es keine Konstante, kein Funktions- oder Relationssymbol gibt, das in beiden Signaturen auftaucht.

Die zweite wichtige Einschränkung, die wir ziehen, betrifft die Lösungsbereiche. Diese müssen quasi-freie Strukturen sein. Der Begriff der quasi-freien Struktur wurde von F. Baader und K. U. Schulz in [10] eingeführt. Er stellt eine Verallgemeinerung des Begriffs der freien Struktur dar und umfaßt viele wichtige nicht-numerische, unendliche Lösungsbereiche. Unter anderem sind Termalgebren, Quotiententermalgebren, rationale Baumalgebren, Vektorräume, erblich endliche fundierte und nicht fundierte Listen, Mengen und Multimengen sowie bestimmte Arten von Feature-Strukturen alle quasi-freie Strukturen. Die wesentliche Erweiterung gegenüber freien Strukturen besteht darin, daß die Elemente des Grundbereiches nicht generiert sein müßen. Es reicht, wenn sie in ihrem Verhalten unter Abbildungen durch das, was mit ihren „Atomen“ geschieht, determiniert sind. Quasi-freie Strukturen sind symbolisch und unendlich. Dies stellt eine Einschränkung insofern dar, daß numerische Bereiche und endliche Bereiche, die beide wichtige Bereiche für Constraint-Löser darstellen, keine quasi-freien Strukturen sind, und also unsere Methoden der Kombination für diese nicht in Frage kommen.

Baader und Schulz stellen in [10, 12, 15] ein erstes allgemeines Kombinationsverfahren vor, das wir eingehend erläutern, da es für unsere Arbeit grundlegend ist. Die kombinierte Lösungsstruktur ist das sogenannte Freie Amalgamierte Produkt. Unter allen sinnvollen kombinierten Strukturen ist es dadurch charakterisiert, die allgemeinste zu sein in dem Sinne, daß ein homomorphes Bild des Freien Amalgamierten Produkts in jeder kombinierten Lösungsstruktur zu finden ist. Die Autoren geben ein allgemeines Verfahren zur Konstruktion des Freien Amalgamierten Produkts für zwei beliebige quasi-freie Strukturen an. Die geforderte Konservativität zeigen sie, indem sie beweisen, daß, betrachtet man jeweils nur eine Komponentensignatur, „vergißt“ also sozusagen im Freien Amalgamierten Produkt die Signatur der anderen Komponente, das Freie Amalgamierte Produkt und die Komponentenstruktur isomorph sind. Die Autoren präsentieren weiterhin einen Dekompositionsalgorithmus, der gemischte Constraint-Probleme in reine Constraint-Probleme der jeweiligen Komponenten zerlegt. Mit Hilfe dieses Dekompositionsalgorithmus beweisen sie, daß die positive Theorie des Freien Amalgamierten Produktes entscheidbar ist, wenn die positiven Theorien der Komponentenstrukturen entscheidbar sind.

Der erste Teil unseres Beitrags setzt sich mit diesem Dekompositionsalgorithmus auseinander und gibt Optimierungsverfahren an. Der Dekompositionsalgorithmus beinhaltet drei nichtdeterministische Schritte, die einen so großen Suchraum aufspannen, daß sich eine Implementation dieses Algorithmus schlicht verbietet. Um also den Dekompositionsalgorithmus auch für Anwendungen brauchbar zu machen, muß man Optimierungsverfahren finden, die den Suchraum drastisch einschränken. Dabei lag unser Interesse darin, möglichst allgemeine Verfahren zu finden, die auf fast alle Constraint-Löser von quasi-freien Strukturen anwendbar sind. Es werden zwei solch allgemeiner Optimierungsverfahren vorgestellt, genannt die deduktive und die iterative Methode. Die iterative Methode wurde für den Fall entwickelt, da eine größere Zahl von Komponenten als nur zwei kombiniert werden. Sie beruht auf der Beobachtung, daß das Dekompositionsverfahren in so einem Fall den Suchraum stark vergrößert, da alle möglichen nicht-deterministischen Entscheidungen getroffen werden, bevor auch nur eine Komponente zu lösen versucht wird. Im Gegensatz dazu arbeitet die iterative Methode lokal. Sie legt die nicht-deterministischen Entscheidungen jeweils nur für eine Komponente fest, und schreitet erst dann zur nächsten fort, wenn ein Satz von Entscheidungen getroffen wurde, für den der Komponentenconstraint-Löser eine Lösung findet. Die Aufgabe bei dieser an sich einfachen Idee besteht darin, zu zeigen, daß das so modifizierte Verfahren unsere Anforderung noch erfüllt, daß es dann und nur dann eine Lösung für ein gemischtes Constraint-Problem im Freien Amalgamierten Produkt gibt, wenn es Lösungen für reine Constraint-Probleme in den Komponenten gibt.

Die deduktive Methode beruht auf der Einsicht, daß nicht alle Entscheidungen im Dekompositionsalgorithmus nicht-deterministisch getroffen werden müssen. Oft stellen das zu lösende Constraint-Problem und die einzelnen Komponenten Vorgaben, wie eine bestimmte Entscheidung zu treffen ist, soll das Problem noch lösbar sein. Man benötigt dazu neue Constraint-Löser für die Komponenten, die am Prozeß der Suche nach den richtigen Entscheidungen beteiligt werden können. Diese müssen für ein ihnen vorgelegtes reines Constraint-Problem dann nicht mehr nur Lösbarkeit feststellen, sondern auch angeben können, welche Entscheidungen im Algorithmus wie zu treffen sind, damit ihr reines Constraint-Problem lösbar bleibt. Der Kombinationsalgorithmus wird bei der deduktiven Methode damit zu einer Art Moderator. Er befragt reihum die beteiligten Komponentenlöser, welche Entscheidungen wie zu treffen sind. Wenn schließlich die Komponentenlöser keine weiteren Entscheidungen festlegen können, trifft der Kombinationsalgorithmus eine nichtdeterministische Entscheidung und beginnt die Konsultation der Komponentenlöser erneut, bis schließlich auf diesem Wege alle Entscheidungen getroffen sind. Sagen dann noch alle Komponentenlöser, daß ihre reinen Constraintprobleme lösbar sind, so ist auch das ursprüngliche gemischte Problem lösbar.

Da die beiden Optimierungsverfahren völlig unabhängig voneinander sind, lassen sie sich in eines integrieren. Die Verfahren sind tatsächlich implementiert, um ihren Effekt auch testen zu können. Von uns durchgeführte Tests zeigen, daß viele Constraint-Probleme mit den Optimierungen um Größenordnungen schneller gelöst werden können, und einige Probleme erst mit dem optimierten

Verfahren praktisch lösbar sind.

Der zweite Beitrag behandelt die sogenannte *Rationale Amalgamierung*. Dabei handelt es sich um eine zweite allgemeine Kombinationsmethode neben dem Freien Amalgamierten Produkt. Obwohl das Freie Amalgamierte Produkt eine allgemeinste kombinierte Lösungsstruktur darstellt, gibt es doch in der Praxis Kombinationen, die nicht Instanzen des Freien Amalgamierten Produktes sind. Beispiele dafür sind Arbeiten von A. Colmerauer [29, 30] über die Kombination von rationalen Bäumen und rationalen Listen, sowie Arbeiten von W. Rounds [90] und A. Moshier & C. Pollard [75] über die Kombination von Feature-Strukturen und nicht-fundierten Mengen. In allen diesen Fällen sind die Komponentenstrukturen „rational". Freie Amalgamierung ist daher nicht die Kombination der Wahl für diese Constraint-Systeme, da die Elemente in der kombinierten Lösungsstruktur nicht rational, sondern endlich sind, das heißt, gemischte Constraints, in denen ein unendlich häufiger Wechsel von der einen Komponente in die andere verlangt wird, sind im Freien Amalgamierten Produkt nicht lösbar.

Dies ist der Ansatzpunkt der Rationalen Amalgamierung. Sie stellt eine kombinierte Lösungsstruktur bereit, in der rationale Elemente vorhanden, mithin solche Constraints lösbar sind. Damit ist Rationale Amalgamierung die Methode der Wahl bei der Kombination von rationalen Komponentenstrukturen. Sie liefert ein allgemeines Konstruktionsverfahren für eine rationale kombinierte Lösungsstruktur, gegeben zwei beliebige sogenannte nicht-kollabierende quasifreie Strukturen. Auch für diese kombinierte Lösungsstruktur läßt sich zeigen, daß bei Einschränkung auf jeweils eine Signatur nur einer Komponente die kombinierte Struktur isomorph ist zur Komponentenstruktur. Damit ist die Forderung nach Konservativität erfüllt. Wir beweisen, daß das Freie Amalgamierte Produkt bis auf Isomorphie eine Teilstruktur der Rationalen Amalgamierung ist. Auch das folgende Theorem zeigt die Natürlichkeit der Konstruktion: Die rationale Amalgamierung zweier rationaler Baumalgebren über disjunkten Signaturen ist isomorph zur rationalen Baumalgebra über der vereinigten Signatur. Der Dekompositionsalgorithmus für Rationale Amalgamierung ist eine vereinfachte Variante des Algorithmus für das Freie Amalgamierte Produkt; der letzte der drei nichtdeterministischen Schritte ist bei Rationaler Amalgamierung nicht erforderlich. Mithilfe dieses Dekompositionsalgorithmus läßt sich zeigen, daß die Lösbarkeit von gemischten Constraints in der Rationalen Amalgamierung entscheidbar ist, wenn die Lösbarkeit reiner Constraints (mit gewissen technischen Zusatzbedingungen) in den Komponentenstrukturen entscheidbar ist. Damit ist ein zweites allgemeines Kombinationsverfahren gegeben.

Der dritte Beitrag befaßt sich mit Negation bei der Kombination von Constraint-Systemen. Die Constraints, die in den bisherigen Teilen betrachtet wurden, waren allesamt positive Constraints, enthielten also weder Negation noch Implikation. Nun ist aber insbesondere die Implikation ein sprachliches Mittel, auf das man bei der Formulierung von Constraint-Problemen ungern verzichtet. Sie wird beispielsweise verwendet, um Constraint-Entailment auszudrücken, das bei der Reduktion von Constraintmengen wichtig ist. In einem ersten Teil setzen wir uns daher mit der Kombination von Constraint-Systemen, deren Sprachen Negation enthalten, auseinander. Dabei gehen wir vom Freien Amalgamierten Produkt als kombinierter Lösungsstruktur aus und verwenden als Algorithmus eine Erweiterung des Dekompositionsalgorithmus um die Behandlung negativer Formeln, aber ohne neue nichtdeterministische Schritte. Wir zeigen, daß das existenzielle Fragment der vereinigten Signatur, also Konjunktionen und Disjunktionen von gemischten Atomformeln und gemischten negierten Atomformeln, im Freien Amalgamierten Produkt entscheidbar ist, wenn die existenziellen Fragmente der Komponenten mit gewissen technischen Zusatzbedingungen entscheidbar sind. Weiterhin geben wir Bedingungen für die Lösungen in den Komponenten an, unter denen sich im Freien Amalgamierten Produkt Grundlösungen für gemischte Constraint-Probleme finden lassen, da bei Constraint-Problemen mit Negation oftmals ein besonderes Interesse an Grundlösungen besteht. Leider kann man aus Arbeiten von R. Treinen [113] erkennen, daß sich im allgemeinen im Freien Amalgamierten Produkt auch dann kein größeres Quantorenfragment als das existenzielle entscheiden läßt, wenn dies in den Komponenten möglich ist. Es ergibt sich außerdem, daß Rationale Amalgamierung für die Kombination von Constraint-Systemen mit Negation höchstens in Spezialfällen geeignet ist, allgemein jedoch nicht.

Im zweiten Teil dieses Beitrags geht es um die sogenannte Unabhängigkeitseigenschaft negativer Constraints. Ein Constraint-System besitzt die Unabhängigkeitseigenschaft, wenn für jede Konjunktion von positiven Constraints und jede Konjunktion von negativen Constraints gilt: Die beiden Konjunktionen sind zusammen lösbar genau dann, wenn für jeden negativen Constraint die Konjunktion positiver Constraints zusammen mit dem negativen Constraint lösbar ist. Man kann also die negativen Constraints unabhängig voneinander lösen. Diese Eigenschaft ist wichtig in praktischen Constraint-Lösern. Sie ermöglicht es, einen Constraint-Löser nur für positive Constraints zu entwickeln und mit diesem dennoch mithilfe einer Übersetzung auch negative Constraints zu lösen. Zuerst einmal ohne Berücksichtigung von Kombinationen untersuchen wir, welche quasi-freien Strukturen die Unabhängigkeitseigenschaft besitzen, wobei wir einen besonderen Blick auf Gleichungstheorien werfen, da diese eine wichtige Rolle unter den quasi-freien Strukturen spielen. Für Gleichungstheorien ergibt sich, daß unitäre Theorien die Unabhängigkeitseigenschaft besitzen, finitäre aber nicht. Es gibt Theorien vom Typ 0, die die Unabhängigkeitseigenschaft besitzen. Für die Kombination von Gleichungstheorien erhält man folgendes Modularitätsresultat: Das Freie Amalgamierte Produkt zweier unitärer, regulärer und kollaps-freier Theorien ist wieder unitär, hat mithin also die Unabhängigkeitseigenschaft. Blickt man nun allgemeiner auf quasi-freie Strukturen, so sieht man, daß auch hier die unitären Strukturen die Unabhängigkeitseigenschaft besitzen. Und es gibt infinitäre Strukturen mit Unabhängigkeitseigenschaft. Zum Abschluß läßt sich auch das Modularitätsresultat verallgemeinern. Das Freie Amalgamierte Produkt zweier unitärer, regulärer und nicht kollabierender quasi-freier Strukturen ist wieder unitär, hat somit die Unabhängigkeitseigenschaft.

Diese Forschungsarbeit entstand im Rahmen eines Kooperationsprojekts an dem Klaus U. Schulz, Franz Baader und Jörn Richts beteiligt waren. Daher wurden einge Resultate in Zusammenarbeit mit ihnen entwickelt. Wann immer dies der Fall ist, ist dies entsprechend im Text vermerkt.

Acknowledgement

This research was carried out at the Centre for Language and Information Processing (Centrum für Informations- und Sprachverarbeitung, CIS) at the Ludwig-Maximilians-University of Munich, and I would like to thank this institution for its support.

Most of all, I would like to thank my advisor Klaus Schulz. Without his help, encouragement, patience, and adhortations, this thesis would have never come to exist. I cannot count the amount of hints and advises he gave and the time and energy he spent to help realise this project.

I am also grateful to our project partners at the University of Aachen, Franz Baader and Jörn Richts who did not only prove to be fine collaborators but also always had an open ear for discussions.

My thank goes also to the German Research Council (Deutsche Forschungsgesellschaft, DFG) for financing this research project as part of their focus project "Deduktion", and to the European Union for providing travelling funds as part of the Esprit project "Constructions in Computational Logic".

Chapter 1

The Context of Combining Constraint Systems

1.1 Constraint Systems

Constraint systems date back as far as to the mid nineteen sixties, when static constraints were used to describe little toy worlds made of simple geometric objects and constraint solvers had to reason whether an object could be on top of another one (e.g., [110]). They gained significantly in interest during the eighties when constraint programming merged with the field of logic programming to constraint logic programming (see [55] for an overview). From the logic programming side, this merge was natural, since constraints had already been present in logic programming: Unification can be seen as implicitely introducing equality constraints over terms. This step was made explicit in the development of Prolog II [29]. Theoretical concepts were proposed by J. Jaffar and J.-L. Lassez in [54]. From there, it is only a small step to introduce all kinds of constraints and to develop constraint solvers for different domains like linear arithmetic over the real numbers, expressions over Boolean algebras, constraints over string domains or feature trees and more. Nowadays, there are several commercial constraint systems and solvers available in a growing market.

There exists another area in which constraint solving is gaining interest, namely automated theorem proving and term rewriting. In automated theorem proving, one is faced with the problem that certain axioms will lead to non-termination, because these axioms are always applicable. Examples are commutativity or associativity of a function symbol. As a solution to this problem, G. Plotkin [85] proposed to integrate these axioms into the deduction engine, more specifically into the unification algorithm. M. Stickel prosecuted this idea presenting resolution modulo a theory in [108]. A similar problem exists in term rewriting, where the above mentioned axioms seen as equations turn out to be non-orientable. For commutativity, this is easily understood: Which side of the axiom should be seen as more complex, which one as reduced, when they are absolutely symmetrical? It is no coincidence, that the solution proposed here by

J.-P. Jouannaud & H. Kirchner [58] as well as L. Bachmair [18] is similar: These non-orientable equations should be handled by the unification process. Unfortunately, equational unification, especially unification modulo associativity and commutativity, which is the one that is used the most, is not completely well behaved in that a simple unification problem can have a huge number of different unifiers. A solution to this difficulty can be the introduction of constraints. Instead of computing and applying all unifiers, one poses the constraint that the unification problem should remain solvable in further computation steps. Systems based on this idea were proposed by H. J. Bürkert [24] and R. Nieuwenhuis & A. Rubio [78, 91] in the field of automated deduction and by C. and H. Kirchner [66] in the field of term rewriting.

All of these areas are faced with the following problem. In principle, a constraint solver can be seen as a search engine. Therefore there is no such thing as a general problem solver, because a "general" domain would be huge and no search in it could ever terminate in acceptable time. A constraint solver can only be efficient, if it is designed for a particular domain, best a richly structured domain, because only the employment of as much knowledge about the specific domain as is at hand avoids the otherwise threatening explosion of the search space. Hence a constraint solver must be domain specific. On the other hand, non-trivial problems are heterogeneous, not restricted to a single domain. If you want to use a constraint system to model some part of the real world, then different aspects of it will be described by constraints over different domains, unless one forces anything into a single view. In automated theorem proving and term rewriting, there are also several different theories involved, if statements to be proven become more complex.

To overcome the problem that constraint solvers must be domain specific while interesting problems are heterogeneous one needs methods to combine the different constraint solvers. Ideally, the heterogeneous constraints should be interpreted in a combined solution domain that shares relevant structural properties with the different component constraint domains involved. Hence one needs a construction method for such a combined solution domain. And there should be a constraint solver for the combined solution domain with the following properties. It should not be a new solver, rather effectively and efficiently reduce mixed constraints to pure constraints of the participating components in order to use their efficient constraint solvers to solve the problem. And this translation of mixed constraints into pure constraints of the different components should take place in such a way that there is a solution for the mixed constraints in the combined domain if and only if the pure constraints have solutions in their respective solution domains. This sketches the general task.

1.2 Combination of Constraint Systems and Computational Linguistics

Constraint systems play an important role in linguistics, too. Classical government and binding theory [27] describes syntactic well-formedness in terms of principles, parameterised by languages, that any well-formed utterance has to fulfil. One can regard these principles as constraints on the grammaticality of syntactic structures. This view gains strength, if one looks at principle based parsing. Parsers for GB-theory [34, 56, 69, 115] are indeed incarnations of constraint solvers, the syntactic structure they calculate for a given input utterance is a solved form of the set of constraints consisting of the string or phonological representation of the utterance and the linguistic principles and parameters.

The branch of linguistics following Head-driven Phrase Structure Grammar [86, 87] is even clearer related to constraint systems. In this theory, linguistic entities are described by means of feature structures which are sets of constraints. And linguistic principles are also expressed by complex feature structures. Parsers for HPSG (see, e.g., [45, 46, 57]) are hence constraint solvers for the domain of feature structures. The constraint set to be solved consists of the linguistic theory and a small feature structure representing the input utterance. And the full linguistic analysis of the utterance in relation to the theory, expressed again as a feature structure, can be regarded as the normal form of the utterance feature structure on the background of the constraints coding the linguistic theory.

It is also in this area that we can find instance of combination in linguistics. Many linguists demand to extend feature structures to include lists and sets or multi sets of feature structures. W. Rounds [90] as well as A. Moshier & C. Pollard [75] propose frameworks for integrating sets and feature structures. We regard these as instances of combination of a feature theory as one component and the theory of lists or (multi-) sets as the second component. Another such instance of combination with feature structures can be found in works by J. Dörre and A. Eisele [39, 40], who consider the integration of disjunctions into feature structures. From our abstract point of view, this exemplifies the combination of feature structures with Boolean algebras. The advantage of our perspective lies in the clear separation of the different component theories involved, feature structures on one side and sets or Boolean algebras on the other. Consequently, certain decidability results about versions of "enriched" feature structures that we difficult to obtain in their original setting, come out much simpler, if one regards the "enriched" feature structures as instances of combination.

On the long run, computational linguistics is not just concerned with syntactic parsing, but with all aspects of natural language processing, and that includes semantic and pragmatic evaluations of expressions and utterances. Since speakers never utter anything involved in their thoughts, but rely on the intelligence and knowledge of their audience, pragmatics leads almost instantly to the fields of reasoning about the world and deduction. In other words, comprehensive computer linguistics systems need deductive components. And in these components, the problems one is faced with contain most naturally bits and pieces of many different theories, since the world to reason about is heterogeneous. Hence, the need for combining different reasoning components and different constraint systems is quite strong in this area. We admit that these theses are ideas on the long term future of language comprehension systems. But we are convinced that these topics will become important. And we see our work as basic research that generally investigates methods and principles available.

1.3 Combination of Constraint Systems

Let us now present a general framework of our work and explain the problems we would like to deal with. A constraint system in our sense consists of three parts: a constraint language, a solution domain, and a constraint solver. The first component of a constraint system is the constrait language. We will exclusively look at first order languages and fragments thereof. Higher order constraint languages are not very relevant, since most properties expressible in them are undecidable; that says, one cannot really do much constraint solving in them.

A solution domain is an algebraic structure which is used to interpret the constraint language. The solution domains we will be looking at are always infinite. This certainly constitutes a restriction, especially since there are many efficient constraint solving techniques for finite domains, but the infiniteness of the domain is required for our type of combination. Still, not every infinite domain will be right for our purposes. We are interested in symbolic, non-numerical solution domains of a fairly general character. Solution domains that fulfil these properties and are suitable for combination are called quasi-free structures. We will explain this notion, that was introduced by F. Baader and K. U. Schulz in [10], in Section 3.2.2. Quasi-free structures generalise free structures and hence comprise solution domains such as term algebras and quotien term algebras. But also vector spaces, rational tree algebras, well-founded and non-wellfounded lists, sets and multi sets are quasi-free structures. This shows that many important non-numerical infinite solution domains are indeed covered.

A constraint solver is an algorithm that decides the constraint language or a fragment of it. In other words, given a formula, the constraint solver states whether the formula holds true in the solution domain. Very often, constraints are just conjunctions of atomic formulae, variables appearing in them are thought of as being implicitely existentially quantified. Other fragments we will consider are the existential fragment – conjunctions and disjunctions of atoms and negated atoms – and the positive theory, which permits arbitrary quantification of conjunctions and disjunctions of atoms.

Given these constraint systems, we are interested in *systematic* ways of combining them. A combination is systematic, if it provides the following. It gives an explicit construction mechanism to gain a combined solution domain given the solution domains of the components. The combined solution domain should again be a quasi-free structure, and it should share relevant structural properties with its components. Secondly, there must be an algorithm that solves "mixed" constraints in the combined solution domain by reducing them to pure constraints in the components in such a way that a mixed constraint is valid in the combined solution domains if and only if the corresponding pure constraints are valid in the component solution domains. We also demand conservativeness in the sense that a pure constraint over the language of just one component is true in the combined solution domain if and only if it is true in the component. Evaluation of pure constraints should not lead to new results.

An important restriction in combination concerns the signatures of the component systems. A signature comprises the sets of constants, function and predicate symbols that are used to construct the constraint language. In the combinations we will look at, we always assume the signatures to be *disjoint*, i.e., to share no constants, function or predicate symbols, even if we do not state so explicitely. This restriction may seem severe, but to date no-one has come up with decent results on combining constraint systems over non-disjoint signatures.

1.4 Combination of Equational Unification Algorithms: A Historic Overview

Equational theories are amongst the most prominent quasi-free structures. And although we gave a rather abstract definition of the research problem, the question of combination has its strongest roots in unification theory. Unification is *the* underlying operation in automated deduction as well as in rewriting. It is as fundamental to them as is calculus for mathematics. Since the field is too large, we will not give here an overview over unification theory or its history. The interested reader is referred to [17] instead. Rather we describe the developments that led to the question of how to combine equational theories and the solutions proposed to this question. Doing so we preassume a certain familiarity with the notions of unification theory. All these notions are explained in Section 2.2.

In his pioneering work on automated deduction, J. A. Robinson [89] describes in 1965 an algorithm designed to efficiently decide the applicability of a deduction rule. This algorithm is syntactic unification. It should be mentioned that unification was already described more than 30 years before in the work by J. Herbrand [50], which unfortunately became forgotten.

The further development of unification was driven by the insight that certain axioms in automated deduction like commutativity ($C = \{f(x,y) = f(y,x)\}$) or associativity ($A = \{f(x, f(y,z)) = f(f(x,y),z)\}$) of a function symbol f lead to non-termination. Similarly, in term rewriting, the very same equations of commutativity or associativity cannot be oriented. In 1972, G. Plotkin [85] proposed to stick these axioms into the unification algorithm, that is not perform syntactic unification, but unification modulo an equational theory. He also gave a unification algorithm modulo commutativity and described a non-terminating procedure for unification modulo associativity.

The problem of giving an algorithm for unification modulo commutativity and associativity, one that is particularly important since most function symbols in automated deduction are commutative *and* associative, was solved independently in 1975 by M. Stickel [106] and M. Livesey & J. Siekmann [67] by showing that this kind of unification can be reduced to the solving of linear Diophantine equations over the non-negative integers. But these algorithms were algorithms for AC-unification with constants. Additional free function symbols in the unification problems could not be handled.

In 1981, M. Stickel [107] presented a general AC-unification algorithm, but could not prove its termination. This termination problem remained open for quite some time until F. Fages [42, 43] was able to solve it in 1984. The difficulty of this problem exposed the importance of more general questions. Given an algorithm for equational unification with constants, is there a general method to extend it into a general equational unification algorithm? And given algorithms for unification with constants in two different equational theories over disjoint signatures, is there a principled way to gain a unification algorithm for the combined equational theory to solve unification problems with mixed terms?

In 1985, C. Kirchner [64, 65] presented a first solution for a special subcase. Both component theories have to be collapse-free and subterm collapse free. An equational theory E is subterm collapse free if no term is ever E-equal to one of its proper subterms. That same year, K. Yelick [117, 118] proposed a combination method that works for collapse-free regular component theories based on a generalisation of Stickel's algorithm. In 1986, A. Herold [52] presented a different method to combine collapse-free regular equational theories. Also in that year, E. Tidén [111] found a combination method for collapse-free equational theories. Another special method was proposed by A. Boudet, J.-P. Jouannaud & M. Schmidt-Schauß [22] in 1989. In their algorithm, only one of the component theories bears restrictions. It has to be so-called cycle-free. An equational theory is cycle-free, if any unification problem of the form $x = t$ where t is a non-variable term and $x \in \mathit{Var}(t)$ has no solution. The second component theory is unrestricted. But one needs a further algorithm to finitely solve so-called constant elimination problems[1].

Finally in 1988, M. Schmidt-Schauß [92, 93] presented a general solution to the combination problem where no restrictions are imposed on the component theories. The requirements are a unification algorithm for unification with constants for both component theories, which entails that the component theories must be finitary; and a constant elimination algorithm for both components. In 1990, A. Boudet [20, 21] described a more efficient version of this combination method.

All of the above described methods require that all occurring unification problems and constant elimination problems have always finite complete sets of solutions in the component theories and actually calculate and use these solutions. Hence they are not suitable, if the component algorithms are decision procedures, i.e., procedures that just state solvability of a given problem and do not compute complete sets of solutions. And for infinitary theories, only decision procedures can be given. An example of such a theory is the theory A of

[1] A constant elimination problem in a theory E is a finite set $\{(c_1, t_1), \ldots, (c_n, t_n)\}$ where the c_i's are free constants not occurring in the signature of E and the t_i's are terms. A solution to such a problem is a substitution σ such that for all i $(1 \le i \le n)$ there exists a term t'_i not containing the constant c_i with $t'_i =_E \sigma(t_i)$.

an associative function symbol. A non-terminating procedure that enumerates all unifiers of a given problem was already presented by G. Plotkin [85] back in 1972. In 1977, G. S. Makanin [71] showed that A-unification with constants is decidable. But the decidability of general A-unification remained an open problem. Based on the ideas of M. Schmidt-Schauß and A. Boudet, and motived by this problem, F. Baader and K. U. Schulz developed a combination method for decision procedures [5, 14] in 1992. They show how to reduce the decidability of combined unification problems to the decidability of pure unification problems with so-called linear constant restrictions in the components. The authors subsequently extended these results to constraint solving. The following year, they showed that this method can be extended to handle combined disunification problems, i.e., equations and disequations between mixed terms [6, 9]. By using algebraic methods, they proved that the positive theory of the combination of equational theories is decidable provided the positive theories of the components are decidable [8]. And they generalised the components from equational theories to quasi-free structures, which comprise many solution domains for constraint solving [10] in 1995.

1.5 Combination of Satisfiability Procedures

There exists a different type of combination of theories in the literature, that we will call here *cooperation* in order to distinguish it from our type of combination. In this view, a theory is a (deductively closed) set of sentences of first order logic. A decision procedure determines whether a universal sentence is satisfiable in that theory, that is to say, whether there exists a model of the theory and the universal sentence. The problem of cooperation of decision procedures then can be described as follows. Given decision procedures for theories over disjoint signatures, is there a general method to construct a decision procedure for the combined theories to decide universal sentences over the joined signature.

The question of cooperation of decision procedures appeared first in the late nineteen seventies in the context of automated program verification. Researchers were confronted with the problem that programs typically contain heterogeneous constructs such as lists or arrays and arithmetical expressions over reals or integers. While decision procedures for fragments of such theories were known, the task was to devise coordination methods to decide mixed formulae. Building upon earlier work on the augmentation of the universal fragment of Presburger arithmetics by uninterpreted predicates and functions [99], R. Shostak developed an integrative cooperation method, which was intendedly not very modular, for theories with normal forms for interpreted predicates and functions in [100].

About the same time, G. Nelson and D. Oppen [76, 77] devised a general method for the cooperation of decision procedures based on the exchange of equations and disequations of variables between the component procedures. Their approach was clearly modular, but rather procedural. D. Oppen [80] offers a more declarative non-deterministic version. Recently, C. Tinelli and M. Harandi [112] gave a simplified proof of the Nelson-Oppen cooperation procedure. C. Ringeissen [88] offers a different perspective and generalises the results to certain non-disjoint combinations.

Although combination and cooperation are related, there are differences to be noted. Cooperation connects decision procedures for *satisfiability*, while combination deals with *validity*. The component procedures in cooperation hence are searching for a model, in combination they test validity in a specific given model. And in cooperation, the combined solution domains are trivial.

In cooperation, one is interested in satisfiability. Consequently, a component decision procedure tries to find a structure that is a model for the theory and for the first-order sentence to be solved. There is no particular interest on the qualities of the model, with the exception that the model should be infinite – at least for the Nelson-Oppen-procedure – to be suitable for cooperation. In combination, there is no search for an arbitrary model. Rather, the model is given, namely a quasi-free structure. The component decision procedure then checks, whether the given (pure) constraint is true in the given structure. The reason is that in combination, one is interested in validity. And quasi-free structures as generalisations of free structures share their property, shown by Mal'cev [73], of being the characteristic model for a theory in the following sense. A positive sentence (or a positive constraint) is valid, i.e., true in all structures of a given theory, if and only if it is true in the quasi-free structure of that theory. Hence validity of a positive constraint can be tested by deciding if it holds true in the quasi-free structure. For negative constraints, this characterisation is no longer valid. But a solution in the quasi-free structure still presents a general solution in some sense. Another important point is that, practically, constraint solvers are not model generators or model seekers. Typically, the solution domain is built-in and hence fixed. What they perform is to check, if a constraint is true in that domain. So combination is more appropriate for constraint solvers, because it assumes that the component solution domains are given, and quasi-free structures comprise many important solution domains.

As explained above, combination does not only deliver an algorithm to reduce the solution of mixed constraints to the solution of pure constraints in the components, but also an explicit construction of a combined solution domain. This combined solution domain, which should share relevant structural properties with its components, is the domain where mixed constraints are interpreted. Since we are interested in validity, the combined solution domain has to be a quasi-free structure, too. This is the reason why we need an explicit construction method. In the case of coordination, the situation is simpler. If there are infinite models for the pure sentences, a combined model can be gained by means of a bijection between their universes, which always exists after the component universes have been brought to equal cardinality by an application of the Löwenheim-Skolem Theorem. A consequence of this fact is that the decomposition algorithm for coordination is also simpler than the one for combination. Indeed, the decomposition algorithm for combination consists of three non-deterministic steps, while the one for coordination contains only the first one of these non-deterministic steps.

1.6 An Outline of the Thesis

The next chapter is a chapter on technical preliminaries introducing concepts and notions from algebra and unification theory that we will make use of later.

Chapter 3 is devoted to the description of quasi-free structures and the free amalgamated product. We quote here work by F. Baader and K. U. Schulz [12, 15] to introduce concepts and notions fundamental to our work. Quasi-free structures are the solution domains we assume in our constraint systems. As generalisations of free structures, they comprise many important non-numerical infinite solution domains such as term algebras and quotient term algebra, rational tree algebras, hereditarily finite well-founded and non-wellfounded lists, sets and multi sets, vector spaces, and certain feature structures. Quasi-free structures have a simple, purely algebraic definition. While in a free structure, every element of the domain must be finitely generated, this is no longer required with quasi-free structures. It suffices that each element has a kind of a "handle" that can be used to determine the image of this element under an arbitrary homomorphism.

The free amalgamated product constitutes a first systematic way to combine constraint systems. It comes equipped with a general method that, given two arbitrary quasi-free structures, constructs a combined solution domain and a decomposition algorithm to solve "mixed" constraints, i.e., constraints built over the language of the joint signature, by reducing them to pure constraints in the components. The free amalgamated product as solution domain is characterised by the important property of being the most general combination of two quasi-free structures in the sense that every other combination contains a homomorphic image of it. To show the above demanded conservativeness, we present the theorem that the reduct of the free amalgamated product to one component signature is isomorphic to the component structure. The decomposition algorithm translates mixed constraints over the joint signature into pure constraints of the component signatures in such a way, that a solution for a mixed constraint exists in the free amalgamated product if and only if there are solutions of the translated pure constraints in the component domains. The core of the algorithm consists of three non-deterministic steps that guess an information exchange about variables occurring in both components' constraints after translation. On the base of this algorithm, Baader and Schulz show that the positive fragment of the constraint language of the joint signature is decidable in the free amalgamated product, provided the positive fragments of the component constraint languages over their signatures are both decidable.

Our first contribution, Chapter 4, examines this decomposition algorithm and provides optimisation techniques therefor. The three non-deterministic steps mentioned above span a search space that huge that in a practical implementation, even small constraint problems are intractable. To overcome this problem, one has to find methods which drastically shrink the search space. Our interest was to find general methods that should be applicable to a wide range of quasi-free structures and their constraint solvers. Two such optimisation method are presented, called the iterative and the deductive method. The iterative method is designed for the combination of more than two constraint solvers; and the higher the number of solvers combined, the more important this method becomes. It is based on the observation, that the original decomposition algorithm largely expands the search space in such a case, because it makes all non-deterministic decisions for all components first, before testing the first component problem for solvability. The iterative method localises non-deterministic choices, it tries to solve a single component first by guessing those non-deterministic choices that are relevant for that component and proceeds to the next component only after it could solve the current one. The difficulty of this simple idea lies in proving that the thus amended decomposition algorithm is still correct and complete, i.e., that the mixed constraints have a solution if and only if the the iterative method finds one for each pure component problem.

The deductive method is based on the insight that not every decision has to be made non-deterministically. Indeed, one can find that in many cases the constraint problem at hand and the component solution domains and constraint solvers determine certain choices in the sense that only if they are made in one particular way, the problem is solvable. To use the deductive method, one needs new component constraint solvers, which have to do more than just decide whether a pure constraint problem is solvable. They must be capable of computing which choices have to be made deterministically in what way in order to keep their component problem solvable. The combination algorithm hence becomes kind of a moderator. It asks the component solvers, which decisions can be made deterministically and in what way, percolating these informations from component solver to component solver. Only after no new decisions can be made deterministically, it picks one non-deterministic choice. Thereafter it starts again consulting the component solvers, and so on, until all decisions are made. If then still all component solvers state their pure constraint problem is solvable, then the original mixed problem has a solution.

Since the two optimisation methods are totally independent of each other, one can easily integrate them into a common system. In order to test the efficiency of the optimisation, we implemented several versions of them. The test results provided show that many problems can be solved several orders of magnitude faster than with the original algorithm by Baader & Schulz. And there are some examples that cannot be solved in reasonable time without the optimisation methods.

The concept of *rational amalgamation* is introduced in Chapter 5. It establishes a second systematic way of combining constraint systems. Although the free amalgamated product is the most general combination method, there are examples of combination that are not instances of the free amalgamated product. Work by A. Colmerauer on the combination of rational tree algebras and rational nested lists [29, 30] and work by W. Rounds [90] and A. Moshier & C. Pollard [75] on the combination of feature structures and non-wellfounded sets are such examples. In all of these cases, the component structures are "rational", they contain infinite (but periodic) elements. In such a situation, the free amalgamated product is not the combination of choice, because the elements in the combined solution domain are always finite. Mixed constraints that demand an infinite number of transitions from one component to the other are therefore unsolvable in the free amalgamated product.

This is where rational amalgamation steps in. It provides a combined solution domain that contains rational elements and hence allows to solve such mixed constraints. Therefore, rational amalgamation is the method of choice when combining constraint domains that are themselves rational. It comes equipped with a general construction method for a rational combined solution domain, given two arbitrary so-called non-collapsing quasi-free structures. For this combined solution domain, we show that the reduct to just one component signature is isomorphic to that component domain ensuring in this way the demanded conservativeness. We proof that the free amalgamated product is – up to isomorphism – a substructure of rational amalgamation. The following theorem shows the naturalness of the construction, too: the rational amalgamation of two rational tree algebras over disjoint signatures is isomorphic to the rational tree algebra of the joint signatures. The decomposition algorithm for rational amalgamation is a simplified variant of the one for the free amalgamated product; the final of the three non-deterministic steps is not required in rational amalgamation. Using this decomposition algorithm, we show that solvability of mixed constraints is decidable in the rational amalgamation, if solvability of pure constraints (with certain technical requirements) is decidable in both component domains. For the special class of rational non-collapsing structures, the positive existential theory of the rational amalgamation is decidable, if the positive universal-existential theory is decidable in both components.

Finally, Chapter 6 deals with negation in combining constraint systems. The constraints we mentioned so far were exclusively positive constraints, they contained neither negation nor implication. But clearly, implication is a desirable tool when formulating constraint problems. For example, constraint entailment cannot be expressed without it. And constraint entailment is used to decide whether one constraint subsumes or entails another when reducing constraints in a constraint solver (see, e.g., the constraint solver Oz [49, 103]). Negation is also an important part of the constraint language in Prolog II and III [29, 30]. Thus we investigate combinations of constraint systems that contain negation in their constraint languages in a first part of this chapter. The combined solution domain we use is again the free amalgamated product, and the algorithm is an extension of the decomposition algorithm to handle negative formulae, but one that does not introduce new non-deterministic steps. We show that the existential theory of the free amalgamated product over the joint signatures is decidable, if the existential theories of both components (with certain technical restrictions) are decidable. Furthermore we give conditions for the solutions in the components under which there exists a ground solution in the free amalgamated product, since there is a special interest in ground solutions, if the constraint problems contain negation. Work by R. Treinen [113] lets us see that in general, it is not possible to solve a larger quantifier fragment in the free amalgamated product than just the existential one, even if one can solve a larger fragment in the components. We also show that Rational Amalgamation is in general a solution domain not suited for the combination of constraint systems with negation.

The second part of this chapter deals with the so-called independence property of negative constraints. A constraint system has the independence property, iff for every conjunction of positive constraints and every conjunction of negative constraints we have: The two conjunctions together are solvable if and only if for each negative constraint the conjunction of positive constraints plus that negative constraint is solvable. In this case, the negative constraints can be solved separately. This property is very important for real world constraint solvers. It enables the development of constraint solvers that solely solve positive constraints and can still handle negative ones by some means of translation. Firstly ignoring combination, we investigate which quasi-free structures own the independence property. Especially we consider equational theories, since they are an important subclass of quasi-free structures. For equational theories, we show that unitary theories have the independence property, while finitary do not. There theories of type 0 that own the independence property. For the combination of equational theories, one gains the following modularity result: The free amalgamated product of two unitary, regular and collapse-free equational theories is again unitary, and hence as the independence property. Generalising to quasi-free structures, we find that unitary structures have the independence property. And there are infinitary structures with the independence property. Finally, we lift the modularity result to the combination of quasi-free structures. The free amalgamated product of two unitary, regular and non-collapsing quasi-free structures is again unitary, and thus has the independence property.

This research was carried out as part of a cooperative project, in which Klaus U. Schulz, Franz Baader, and Jörn Richts participated. Some of the results were therefore developed in cooperation with them. Whenever this is the case, their contributions are clearly marked in the text.

Chapter 2

Preliminaries

In this chapter, we introduce basic notions of algebra and unification theory that we will make frequent use of. Structures and homomorphisms are our fundamental tools, they are as substantial to us as hammer and nails are to a carpenter. Notions from unification theory are important, even if we do not always explicitely use them, because most definitions for quasi-free structures are generalisations from unification theory.

2.1 Algebra

A *signature* Σ consists of a finite set Σ_F of function symbols and a finite set Σ_P of predicate symbols, each with a fixed arity. We assume that equality "=" is a logical constant that does not occur in Σ_P, and which is always interpreted as identity. Given a signature Σ and a countably infinite set of variables X, the set of Σ-terms over X, written $T(\Sigma, X)$, is defined as follows: Every variable $x \in X$ and every constant $c \in \Sigma$ is a term; if $f \in \Sigma_F$ is an n-place function symbol and $t_1, \dots, t_n$ are terms, then so is $f(t_1, \dots, t_n)$. An atomic Σ-formula is an equation $s = t$ between Σ-terms s, t or a relational formula $p(s_1, \dots, s_m)$ where p is an m-ary predicate symbol in Σ_P and $s_1, \dots, s_m$ are Σ-terms. A *literal* is an atomic formula or its negation. A Σ*-formula* is defined as follows: Every literal is a formula; if φ and ψ is a formula and $x \in X$ a variable, then $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \exists x \varphi, \forall x \varphi$ are formulas. A Σ*-matrix* is a quantifier-free formula. A formula is called *positive*, if it contains no negation. A formula φ is *negative*, if it is equivalent to $\neg\psi$ where ψ is a positive formula. A sentence is a formula without free variables. The notion $t(v_1, \dots, v_n)$ (resp. $\varphi(v_1, \dots, v_n)$) indicates that the set of all free variables of the term t (of the formula φ) forms a subset of $\{v_1, \dots, v_n\}$.

A Σ*-structure* $\mathfrak{A}^\Sigma$ has a non-empty carrier set A, and it interprets each $f \in \Sigma_F$ of arity n as an n-ary (total) function $f_{\mathfrak{A}}$ on A and each $p \in \Sigma_P$ of arity m as an m-ary relation $p_{\mathfrak{A}}$ on A. Whenever we use a Roman letter like A and an expression $\mathfrak{A}^\Sigma$ in the same context, the former symbol denotes the carrier set of the Σ-structure denoted by the later expression. Sometimes we will consider several signatures simultaneously. If Δ is a subset of the signature Σ, then any Σ-structure $\mathfrak{A}^\Sigma$ can be considered as a Δ-structure (called the Δ-reduct of $\mathfrak{A}^\Sigma$) by just forgetting the interpretation of the additional symbols. In this situation, $\mathfrak{A}^\Delta$ denotes the Δ-reduct of $\mathfrak{A}^\Sigma$. If the signature Σ contains only function symbols, then the Σ-structure $\mathfrak{A}^\Sigma$ is called a Σ-*algebra*.

We write $\mathfrak{A}^\Sigma \models \varphi(a_1, \ldots, a_n)$ to express that the formula $\varphi(v_1, \ldots, v_n)$ is valid in $\mathfrak{A}^\Sigma$ under the evaluation that maps v_i to $a_i \in A$ $(1 \leq i \leq n)$. Expressions $\vec{a}$ denote finite sequences $a_1, \ldots, a_k$ of elements in A. In order to simplify notation, we will sometimes use $\vec{a}$ also to denote the set $\{a_1, \ldots, a_k\}$.

A Σ-*homomorphism* between Σ-structures $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Sigma$ is a mapping $h : A \to B$ such that

$$\begin{aligned} h(f_{\mathfrak{A}}(a_1, \ldots, a_n)) &= f_{\mathfrak{B}}(h(a_1), \ldots, h(a_n)) \\ p_{\mathfrak{A}}(a_1, \ldots, a_n) &\Longrightarrow p_{\mathfrak{B}}(a_1, \ldots, a_n) \end{aligned}$$

for all $f \in \Sigma_F$, $p \in \Sigma_P$ and $a_1, \ldots, a_n \in A$. Letters $h, g, \ldots$ denote homomorphisms and hom^Σ_{A-B} denotes the set of all Σ-homomorphisms between $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Sigma$. In order to increase readability, we will often use expressions of the form h^Σ_{A-B} to denote an element of hom^Σ_{A-B}. A Σ-*endomorphism* of $\mathfrak{A}^\Sigma$ is a homomorphism $h : \mathfrak{A}^\Sigma \to \mathfrak{A}^\Sigma$. With $End^\Sigma_{\mathfrak{A}}$ we denote the monoid of all endomorphisms of the Σ-structure $\mathfrak{A}^\Sigma$, with composition as operation. A Σ-*isomorphism* is a bijective Σ-homomorphism $h : \mathfrak{A}^\Sigma \to \mathfrak{B}^\Sigma$ such that

$$p_{\mathfrak{A}}(a_1, \ldots, a_n) \Longleftrightarrow p_{\mathfrak{B}}(h(a_1), \ldots, h(a_n))$$

for all $p \in \Sigma_P$ and all $a_1, \ldots, a_n \in A$. Equivalently, one can require that the inverse mapping h^{-1} is also homomorphic. A Σ-*automorphism* is an isomorphic Σ-endomorphism.

If $g : A \to B$ and $h : B \to C$ are mappings, then $h \circ g : A \to C$ denotes their composition. When composing several functions, we sometimes drop the $\circ$ and write hgf for $h \circ g \circ f$. Let $g_1 : A \to C$ and $g_2 : B \to D$ be two mappings, We say that g_1 and g_2 coincide on $E \subseteq A \cap B$, iff $g_1(e) = g_2(e)$ for all $e \in E$. For a set A we denote the identity mapping on A by id_A. If A is the carrier of the Σ-structure $\mathfrak{A}$, then id_A is the unit of the monoid $End^\Sigma_{\mathfrak{A}}$.

There is an interesting connection between surjective homomorphisms and positive formulae, which will be important in many subsequent correctness proofs (see [73], pp. 143f or [68] for a proof).

Lemma 2.1.1 *Let $h : \mathfrak{A}^\Sigma \to \mathfrak{B}^\Sigma$ be a surjective homomorphism between Σ-structures $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Sigma$, $\varphi(v_1, \ldots, v_m)$ be a positive Σ formula, and $a_1, \ldots, a_m$ be elements of A. Then $\mathfrak{A}^\Sigma \models \varphi(a_1, \ldots, a_m)$ implies $\mathfrak{B}^\Sigma \models \varphi(h(a_1), \ldots, h(a_m))$.*

Expressions like $\vec{v}, \vec{a}$ are used to denote finite sequences. If $\vec{a} = a_1, \ldots, a_n$ is a sequence of elements of A and if m is a mapping with domain A, then $m(\vec{a})$ denotes the sequence $m(a_1), \ldots, m(a_n)$. If $\vec{v} = v_1, \ldots, v_n$, then $\mathfrak{A}^\Sigma \models \varphi(\vec{v}/\vec{a})$ is shorthand for $\mathfrak{A}^\Sigma \models \varphi(v_1/a_1, \ldots, v_n/a_n)$. The symbol "$\uplus$" denotes disjoint set union. With $|A|$ we denote the cardinality of the set A. If f is a function and M a set, we define the application of f onto M as $f(M) := \{f(m) \mid m \in M\}$.

2.2 Unification Theory

The aim of this section is to introduce those parts and notions of unification theory that are necessary to understand the text. Thus this is not supposed to be an introduction to unification theory in general. An excellent such introduction is [17], from which we take most technical definitions.

In unification theory, a signature Σ consits only of a finite set of function symbols; there are no predicate symbols involved apart from equality. With $\mathcal{T}(\Sigma, X)$ we denote the free term algebra of signature Σ generated by X. A Σ-substitution σ is an endomorphism of $\mathcal{T}(\Sigma, X)$ where the set $\{x \in X \mid \sigma(x) \neq x\}$ is finite. We call this set the domain of σ. If t is a term, then $\mathit{Var}(t)$ denotes the set of variables occurring in t.

An *equational theory* over the signature Σ is a set E of equations between Σ-terms. These equations are implicitly universally quantified and represent an axiom system. With $=_E$ we denote the least congruence relation on $\mathcal{T}(\Sigma, X)$ that is closed under substitution and contains E; and $\mathcal{T}(\Sigma, X)/{=_E}$ denotes the quotient term algebra modulo $=_E$. An equational theory E is called *consistent*, if $x \neq_E y$ for distinct variables $x, y \in X$. E is called *collapse-free*, if E does not contain an equation of the form $x = t$ where $x \in X$ is a variable and t a non-variable term. E is *regular*, if $\mathit{Var}(s) = \mathit{Var}(t)$ for each equation $s = t$ in E. E is *simple*, if $s \neq_E t$ for all terms s, t where s is a proper subterm of t.

Let E be an equational theory with signature Σ. A unification problem is a finite set of equations between terms $\Gamma = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$. A substitution σ is a solution of Γ (also callled a unifier of Γ), iff for all $i = 1, \ldots, n$ holds that $\sigma(s_i) =_E \sigma(t_i)$. Let σ and τ be two solutions of Γ. Then σ is called *more general* than τ, if there exists a substitution λ with $\mathit{Var}(\Gamma)$ as domain such that $\tau =_E \lambda \circ \sigma$. In this case we say that τ is an instance of σ or more specific than σ and write $\sigma \leq \tau$. A substitution μ is called most general unifier of Γ, iff each unifier of Γ is an instance of μ. A set S of solutions of a unification problem Γ is called *complete*, iff for every solution τ of Γ there exists a solution $\sigma \in S$ such that τ is an instance of σ. A solution set S is *minimal*, iff for each two substitutions $\sigma, \sigma' \in S$ neither $\sigma \leq \sigma'$ nor $\sigma' \leq \sigma$ holds. Obviously, a compact representation of all solutions for a unification problem in terms of a minimal complete set is desireable. But unfortunately, such a set may not always exist. Indeed, one systematically divides equational theories by looking at these sets. An equational theory E is called

unitary, iff each solvable unification problem has one most general unifier.
The best known example of such a theory is of course syntactic (or Robinson) unification, an other example is distributivity to the left (or to the right): $D_L = \{f(g(x, y), z) = g(f(x, z), f(y, z))\}$.

finitary, iff each solvable unification problem has a finite minimal complete set of unifiers.
Amongst examples of such theories are commutative unification ($C =$

$\{f(x,y) = f(y,x)\}$) and associative-commutative unification ($AC = C \cup \{f(f(x,y),z) = f(x, f(y,z))\}$).

infinitary, iff each solvable unification problem has a minimal complete set of unifiers, but for at least one problem this set is infinite.
The theory of associative unification, sometimes called word unification, ($A = \{f(f(x,y),z) = f(x, f(y,z))\}$) is an example.

of type 0, iff there exists a solvable unification problem that has no minimal complete set of unifiers.
The theory of idempotent associative unification (also called bands, $AI = A \cup \{f(x,x) = x\}$) is of type 0.

Let E be an equational theory and Γ a set of terms. If it exists, we write $\mu U_E(\Gamma)$ for the minimal complete set of unifiers of Γ, or just $\mu U(\Gamma)$ in case E is clear from context.

Unification problems are systematically distinguished based on what constructors are allowed for building the problem terms. Let E be an equational theory with signature Σ. An *elementary E-unification problem* is a finite set Γ of equations between Σ-terms. In an *E-unification problem with constants*, the terms of the equations may also contain additional constant symbols not contained in Σ. An E-unification problem is called *general*, if the terms may contain additional free function symbols. Note that every general E-unification problem can be regarded as an elementary unification problem in the combined theory $E \cup F$ where F is the free theory over a suitable set of function symbols. The following facts (taken from [17]) show that it is indeed important to distinguish these different types of unification.

- There exists a theory, namely the theory of Abelian monoids ($AC1 := AC \cup \{f(x,1) = x\}$), which is unitary with respect to elementary unification, but only finitary with respect to unification with constants.
- There exists an equational theory for which elementary unification is decidable, but unification with constants is undecidable.
- The theory of Boolean Rings is unitary with respect to unification with constants, but only finitary with respect to general unification.

In later chapters, we will make use of a specific variant of unification with constants. An *E-unification problem with constant restrictions* is an ordinary E-unification problem with constants, Γ, where a set of variables V_c is defined for each free constant c in Γ. A solution of the problem is an E-unifier σ where a free constant c may not occur in $\sigma(x)$ for all $x \in V_c$. The problem Γ is called an *E-unification problem with linear constant restrictions (LCR)* if the sets V_c can be defined by a linear ordering $<$ on the variables and free constants in Γ by $V_c := \{x \mid x \text{ is a variable with } x < c\}$.

Chapter 3

Quasi-free Structures and the Free Amalgamated Product

3.1 Introduction

This chapter introduces the notions of free and quasi-free structures, discusses in general their combinations, presents a particular combination, the free amalgamated product, and algorithms to solve mixed constraints in the free amalgamated product. All of these concepts were developed by F. Baader and K. U. Schulz in [10, 12, 15], they are fundamental to understand our own contributions following in later chapters.

Quasi-free structures are *the* solution domains for combinations. They are generalisations of free algebras and free structures. In a free structure, every element of its domain is finitely generated; it is the result of a finite number of applications of the structure's functions to variables, the structure's generators. These generators have nice properties. Every element of the domain is constructed out of finitely many of them by means of function application. And in order to understand what an arbitrary homomorphism maps an element to, it is sufficient to know what this homomorphism maps its generators to. It is this particular property, that is relevant in combination. The fact that the element is finitely generated, is not so important. Hence the notion of generation will be replaced by the more general one of "stabilisation". We demand the structure to have a distinguished set, to be called atom set, such that every mapping of that set into the carrier of another structure can be extended to a homomorphism into the structure, and each element is stabilised by a finite number of atoms, i.e., if we know where a homomorphism maps these stabilising atoms to, we already know where the element is mapped to. Quasi-free structures comprise most non-numerical infinite solution domains for constraint solving. Term algebras, quotient term algebras, rational tree algebras, vector spaces, hereditarily finite well-founded and non-wellfounded sets and multi sets, hereditarily finite well-founded and non-wellfounded lists, and certain feature structures are all examples of quasi-free structures.

We proceed to combinations of quasi-free structures discussing first fairly generally which conditions have to be fulfilled in order to call a combination suitable. Certainly, not every combination is suitable. One can regard the trivial structure of just one element in the carrier and trivial interpretation of functions and relations as a combination. But that is clearly not our choice. We want the combined structure to be rather general and to share relevant structural properties with its components. We define the notion of the *free amalgamated product* which is characterised as being the most general combination of structures. We also provide a concrete construction method for the free amalgamated product of two quasi-free structures.

Finally we discuss the combination of constraint solvers for quasi-free structures. We present a decomposition algorithm that reduces solvability of mixed constraints over the joint signatures to solvability of pure constraints in the components. The core of the algorithm consists of three non-deterministic steps which together guess the way shared variables of the constraints have to be dealt with in the components in order to ensure that solutions in the components exist if and only if the mixed input problem has a solution. The main theorem reads: The positive theory of the free amalgamated product of two quasi-free structures is decidable, provided the positive theories are decidable in both component structures.

Allmost all parts that follow in this chapter are direct quotes or transcriptions of notions and results presented in [10, 12, 15]. This is also the reason why this chapter contains no proofs. They can all be found in [15]. We decided to quote such extensively from these papers, because we will frequently need the notions and results introduced there. They are really fundamental to our own contributions, there is no way of understanding our work without a knowledge of them.

3.2 Free and Quasi-free Structures

3.2.1 Free Structures

The algebraic theory of free structures is very similar to the one for free algebras, though considerably less well-known. In the first subsection, we will briefly recall some definitions and results for free structures (see [28, 47, 72, 116] for more information). The usual definition of free structures is *external* in the sense that it refers to a whole class of structures. In the present context (i.e., combination of structures and constraint solvers), a characterisation of free structures in terms of their *internal* algebraic structure turns out to be more appropriate. An internal characterisation of free structures over countably infinite sets of generators will be used as starting point for the definition of quasi-free structures in the second subsection. In the third and fourth subsection, we derive useful algebraic and logical properties of quasi-free structures.

We start with the usual external characterisation of free structures.

Definition 3.2.1 Let $\mathcal{K}$ be a class of Σ-structures, and let $\mathfrak{A}^\Sigma \in \mathcal{K}$ be generated by the set $X \subseteq A$. Then $\mathfrak{A}^\Sigma$ is called *free in $\mathcal{K}$ over X* iff every mapping from X into the carrier of a structure $\mathfrak{B}^\Sigma \in \mathcal{K}$ can be extended to a Σ-homomorphism of $\mathfrak{A}^\Sigma$ into $\mathfrak{B}^\Sigma$.[1]

If $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Sigma$ are free in the same class $\mathcal{K}$, and if their sets of generators have the same cardinality, then $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Sigma$ are isomorphic. As shown by the next theorem, it is not really necessary to allow for arbitrary classes of Σ-structures in the definition of free structures. One can restrict the attention to varieties or to the singleton class consisting of the free structure. As for the case of algebras, *Σ-varieties* are defined as classes of Σ-structures that are closed under direct products, substructures, and homomorphic images.

Theorem 3.2.2 *Let $\mathfrak{A}^\Sigma$ be a Σ-structure that is generated by X. Then the following conditions are equivalent:*

1. *$\mathfrak{A}^\Sigma$ is free over X in some class $\mathcal{K}$ of Σ-structures.*
2. *$\mathfrak{A}^\Sigma$ is free over X in some Σ-variety.*
3. *$\mathfrak{A}^\Sigma$ is free over X in $\{\mathfrak{A}^\Sigma\}$.*

The only non-trivial part of the proof, namely "$1 \to 2$", follows from the fact that an algebra that is free in a class $\mathcal{K}$ is also free in the variety generated by $\mathcal{K}$, i.e., the closure of $\mathcal{K}$ under building direct products, substructures, and homomorphic images (see [28, 72] for details).

The third condition of the theorem gives a characterisation of free structures that is independent of any other structure. This motivates the next definition.

Definition 3.2.3 A Σ-structure $\mathfrak{A}^\Sigma$ is called *free* iff it is free over X in $\{\mathfrak{A}^\Sigma\}$ for some subset X of A.

If X is the chosen set of generators of the free structure $\mathfrak{A}^\Sigma$, then we will sometimes indicate this by saying that $(\mathfrak{A}^\Sigma, X)$ is free. We can now give the promised internal characterisation of free structures over countably infinite sets of generators.

Theorem 3.2.4 *A Σ-structure $\mathfrak{A}^\Sigma$ is free over the countably infinite set X iff*

1. *$\mathfrak{A}^\Sigma$ is generated (as a Σ_F-algebra) by X,*
2. *for every finite subset X_0 of X, every mapping $h_0 : X_0 \to A$ can be extended to a* surjective *endomorphism of $\mathfrak{A}^\Sigma$.*

[1] Since $\mathfrak{A}^\Sigma$ is generated by X, this homomorphism is unique.

If one is interested in the question how free structures can be constructed, the characterisation via varieties is more appropriate. We have seen in Theorem 3.2.2 that every free structure is free for some variety. Conversely, it can be shown that every non-trivial variety contains free structures with sets of generators of arbitrary cardinality [72]. The well-known Birkhoff Theorem says that a class of Σ_F-algebras is a variety iff it is an equational class, i.e., the class of models of a set of equations. For structures, a similar characterisation is possible [72].

Theorem 3.2.5 *A class $\mathcal{V}$ of Σ-structures is a Σ-variety if, and only if, there exists a set G of atomic Σ-formulae*[2] *such that $\mathcal{V}$ is the class of models of G.*

In this situation, we say that $\mathcal{V}$ is the Σ-variety defined by G, and we write $\mathcal{V} = \mathcal{V}(G)$.

A concrete description of free Σ-structures can be obtained as follows (see [72, 116] for more information). Obviously, the Σ_F-reduct of a free Σ-structure $\mathfrak{A}^\Sigma$ is a free Σ_F-algebra, and thus it is (isomorphic to) an E-free Σ_F-algebra $\mathcal{T}(\Sigma_F, X)/_{=_E}$ for an equational theory E. In particular, the $=_E$-equivalence classes $[s]$ of Σ_F-terms s constitute the carrier of $\mathfrak{A}^\Sigma$. It remains to be shown how the predicate symbols are interpreted on this carrier. Since $\mathfrak{A}^\Sigma$ is free over X, any mapping from X into $T(\Sigma_F, X)/_{=_E}$ can be extended to a Σ-endomorphism of $\mathfrak{A}^\Sigma$. This, together with the definition of homomorphisms of structures, shows that the interpretation of the predicates must be closed under substitution, i.e., for all $p \in \Sigma_P$, all substitutions σ, and all terms $s_1, \ldots, s_m$, if $p[[s_1], \ldots, [s_m]]$ holds in $\mathfrak{A}^\Sigma$ then $p[[s_1\sigma], \ldots, [s_m\sigma]]$ must also hold in $\mathfrak{A}^\Sigma$. Conversely, it is easy to see that any extension of the Σ_F-algebra $\mathcal{T}(\Sigma_F, X)/_{=_E}$ to a Σ-structure that satisfies this property is a free Σ-structure over X.

Example 3.2.6 Let Σ_F be an arbitrary set of function symbols, and assume that Σ_P consists of a single binary predicate symbol $\leq$. Consider the (absolutely free) term algebra $\mathcal{T}(\Sigma_F, X)$. We can extend this algebra to a Σ-structure by interpreting $\leq$ as subterm ordering. Another possibility would be to take a reduction ordering [37] such as the lexicographic path ordering. In both cases, we have closure under substitution, which means that we obtain a free Σ-structure. Constraints involving the subterm ordering or reduction orderings are, for example, important in constraint rewriting [66].

Free structures over countably infinite sets of generators are canonical for the positive theory of their variety in the following sense:

Theorem 3.2.7 *Let $\mathfrak{A}^\Sigma$ be free over the countably infinite set X in the Σ-variety $\mathcal{V}(G)$, and let φ be a positive Σ-formula. Then the following are equivalent:*

[2] As usual, open formulae are here considered as implicitly universally quantified.

1. *φ is valid in all elements of $\mathcal{V}(G)$, i.e., φ is a logical consequence of the set of atomic formulae G.*

2. *φ is valid in $\mathfrak{A}^\Sigma$.*

This theorem explains why it is appropriate to use free structures over countably infinite sets of generators as solution structures when solving positive constraints. The proof is a simple consequence of Lemma 2.1.1.

We close this subsection by introducing one more definition. If $(\mathfrak{A}^\Sigma, X)$ is free *in* a class of Σ-structures $\mathcal{K}$, then, by definition, $\mathfrak{A}^\Sigma \in \mathcal{K}$. Some authors (see, e.g., [74]) do not assume $\mathfrak{A}^\Sigma \in \mathcal{K}$ when defining the notion "free *for* $\mathcal{K}$." We make use of this less restrictive way of defining "free *for* $\mathcal{K}$" in the following situation:

Definition 3.2.8 Let $\mathfrak{A}^\Sigma$ and $\mathfrak{D}^\Sigma$ be Σ-structures, and assume that $X \subseteq A$ generates $\mathfrak{A}^\Sigma$. $(\mathfrak{A}^\Sigma, X)$ is called *free for* $\mathfrak{D}^\Sigma$ if every mapping $X \to D$ has a unique extension to a homomorphism $h_{A-D} \in Hom^\Sigma_{A-D}$.

3.2.2 Quasi-free Structures

In this section, we generalise the definition of free structures in order to capture typical domains for constraint-based reasoning such as the algebra of rational trees. As illustrating and motivating example for the abstract definitions, we will use free algebras (i.e., free structures where the relational part Σ_R of the signature is empty). In the sequel, let $\mathcal{T} := \mathcal{T}(\Sigma_F, V)/_{=_E}$ be such an algebra (i.e., $\mathcal{T}$ is free over X in the variety defined by the equational theory E, where X consists of the $=_E$-equivalence classes of variables).

Consider an element $[t]$ of $\mathcal{T}$, i.e., the $=_E$-equivalence class of a term t. Obviously, t contains only finitely many variables $v_1, \ldots, v_n$, which shows that $[t]$ is generated by the finite subset $[v_1], \ldots, [v_n]$ of X. Thus, the image of $[t]$ under an endomorphism of $\mathcal{T}$ is determined by the images of the generators $[v_1], \ldots, [v_n]$. In particular, two endomorphisms of $\mathcal{T}$ that coincide on $[v_1], \ldots, [v_n]$ also coincide on $[t]$.

When looking at non-free structures that are used as solution structures for symbolic constraint, one observes that they satisfy algebraic properties that are very similar to those of free algebras. For example, consider the algebra of rational trees where leaves are labelled by constants or variables. This algebra is not generated by the set of variables (since "generated by" talks about a finite process whereas rational trees may be infinite). Nevertheless, a rational tree t contains only a finite number of variables $v_1, \ldots, v_n$, and two endomorphisms of this algebra that coincide on these variables also coincide on t. This means that the variables occurring in rational trees play a role that is similar to the role of generators in free algebras, even though they do not generate the algebra. This observation motivates the definition of stable hulls and atom sets given below.

Definition 3.2.9 Let A_0, A_1 be subsets of the Σ-structure $\mathfrak{A}^\Sigma$. Then A_0 *stabilises* A_1, iff all elements h_1 and h_2 of $End^\Sigma_{\mathfrak{A}}$ that coincide on A_0 also coincide on A_1. For $A_0 \subseteq A$ the *stable hull of* A_0 is the set

$$\mathrm{SH}^{\mathfrak{A}}_\Sigma(A_0) := \{a \in A \mid A_0 \text{ stabilises } \{a\}\}.$$

The following two lemmata show that the stable hull of a set A_0 has properties that are similar to those of the subalgebra generated by A_0. Note, however, that the stable hull can be larger than the generated subalgebra (see Example 3.2.17).

Lemma 3.2.10 *Let A_0 be a subset of the carrier A of $\mathfrak{A}^\Sigma$. Then $\mathrm{SH}^{\mathfrak{A}}_\Sigma(A_0)$ is a Σ-substructure of $\mathfrak{A}^\Sigma$, and $A_0 \subseteq \mathrm{SH}^{\mathfrak{A}}_\Sigma(A_0)$.*

Lemma 3.2.11 *Let A_0, A_1 be subsets of the Σ-structure $\mathfrak{A}^\Sigma$, and let $h \in End^\Sigma_{\mathfrak{A}}$. If $h(A_0) \subseteq \mathrm{SH}^{\mathfrak{A}}_\Sigma(A_1)$, then $h(\mathrm{SH}^{\mathfrak{A}}_\Sigma(A_0)) \subseteq \mathrm{SH}^{\mathfrak{A}}_\Sigma(A_1)$.*

Definition 3.2.12 The set $X \subseteq A$ is an *atom set for* $\mathfrak{A}^\Sigma$ if every mapping $X \to A$ can be extended to an endomorphism of $\mathfrak{A}^\Sigma$.

For the free algebra $\mathcal{T}$ generated by X, the set of generators X obviously is an atom set. Two subalgebras generated by subsets X_0, X_1 of X of the same cardinality are isomorphic. The same holds for atom sets and their stable hulls.

Lemma 3.2.13 *Let X_0, X_1 be atom sets of $\mathfrak{A}^\Sigma$ of the same cardinality. Then every bijection $h_0 : X_0 \to X_1$ can be extended to an isomorphism between $\mathrm{SH}^{\mathfrak{A}}_\Sigma(X_0)$ and $\mathrm{SH}^{\mathfrak{A}}_\Sigma(X_1)$.*

We are now ready to introduce the main concept of this section.

Definition 3.2.14 A countably infinite Σ-structure $\mathfrak{A}^\Sigma$ is called *quasi-free* iff $\mathfrak{A}^\Sigma$ has an infinite atom set X where each $a \in A$ is stabilised by a finite subset of X. We denote this quasi-free structure by $(\mathfrak{A}^\Sigma, X)$.

This definition generalises the characterisation of free structures given in Theorem 3.2.4. The countably infinite set of generators is replaced by a countably infinite atom sets, but we retain some of the properties of generators. In the free case, every element of the structure is generated by a finite set of generators, whereas in the quasi-free case it is stabilised by a finite set of atoms. The following lemma shows that the second condition of Theorem 3.2.4 is satisfied in the quasi-free case.

Lemma 3.2.15 *Let X be an infinite atom set of the countably infinite Σ-structure $\mathfrak{A}^\Sigma$, and let $X_0 \subseteq X$ be finite. Then every mapping $h_0 : X_0 \to A$ can be extended to a surjective endomorphism of $\mathfrak{A}^\Sigma$.*

Remark 3.2.16 Let A^Σ be a Σ-structure and $\mathcal{M}$ be a submonoid of $End_{\mathfrak{A}}^{\Sigma}$. We obtain useful variants of the notions of "stabiliser", "stable hull", "atom set", and "quasi-free structure" by always referring to $\mathcal{M}$ instead of $End_{\mathfrak{A}}^{\Sigma}$. For example, $X \subseteq A$ is an *atom set for* $\mathfrak{A}^\Sigma$ *w.r.t.* $\mathcal{M}$ if every mapping $X \to A$ can be extended to an endomorphism in $\mathcal{M}$. We say that $(\mathfrak{A}^\Sigma, X)$ is *quasi-free with respect to* $\mathcal{M}$ if $(\mathfrak{A}^\Sigma, X)$ satisfies the corresponding variant of Definition 3.2.14. In [10], such structures were called simply combinable structures (SC-structures). Most of the results that we will present for quasi-free structures can be lifted to structures $\mathfrak{A}^\Sigma$ that are quasi-free with respect to some submonoid $\mathcal{M}$ of $End_{\mathfrak{A}}^{\Sigma}$ (see [10] for details).

We will call both $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{A}^\Sigma, X, \mathcal{M})$ quasi-free structures. We think this does no harm, because whenever we refer to a specific submonoid $\mathcal{M}$ of $End_{\mathfrak{A}}^{\Sigma}$, we will make that explicit. In this and the next chapter, $\mathcal{M}$ will always be $End_{\mathfrak{A}}^{\Sigma}$, the set of all endomorphism. In the chapter on rational amalgamation, we do not expect that $(\mathfrak{A}^\Sigma, X)$ is quasi-free with respect to the whole of $End_{\mathfrak{A}}^{\Sigma}$. Thus the quasi-free structures we consider there will have an explicit endomorphism monoid $\mathcal{M}$. In the chapter on negation, we assume that $(\mathfrak{A}^\Sigma, X)$ is quasi-free with respect to $End_{\mathfrak{A}}^{\Sigma}$.

Examples 3.2.17 The following examples show that many solution domains for symbolic constraints are indeed quasi-free structures.

Free structures. Obviously, every free structures over a countably infinite set of generators is a quasi-free structure. The atom set is the set of generators of the free structure.

Vector spaces. Let K be a field, let $\Sigma_K := \{+\} \cup \{s_k \mid k \in K\}$. The K-vector space spanned by a countably infinite basis X is a quasi-free structure over the atom set X. Here "+" is interpreted as addition of vectors, and s_k denotes scalar multiplication with $k \in K$.

The algebra of rational trees. Let Σ_F be a finite set of function symbols, and let $\mathfrak{R}^{\Sigma_F}$ be the algebra of rational trees ([29, 70]), where leaves are labelled with constants from Σ_F or with variables from the countably infinite set V. It is easy to see that every mapping $V \to R$ can be extended to a unique endomorphism of $\mathfrak{R}^{\Sigma_F}$, and that $(\mathfrak{R}^{\Sigma_F}, V)$ is a quasi-free structure. Note, however, that $\mathfrak{R}^{\Sigma_F}$ is not generated by V. In addition, it is easy to see that $\mathfrak{R}^{\Sigma_F}$ cannot be a free structure (over any set of generators). Indeed, it is well-known that only trivial equations between Σ_F-terms are valid in $\mathfrak{R}^{\Sigma_F}$. Thus, if $\mathfrak{R}^{\Sigma_F}$ was free, it would be isomorphic to the absolutely free term algebra, which is not true, however [70].

Hereditarily finite sets. Let $V_{\mathrm{hfs}}(Y)$ be the set of all nested, hereditarily finite (standard, i.e., wellfounded) sets over the countably infinite set of "urelements" Y. Thus, each $M \in V_{\mathrm{hfs}}(Y)$ is finite, and the elements of M are either in Y or in $V_{\mathrm{hfs}}(Y)$, the same holds for elements of elements etc. Wellfounded means that there are no infinite descending membership sequences. Since union is not defined for the urelements $y \in Y$, the urelements will not be treated as sets here. Let $X := \{\{y\} \mid y \in Y\}$. Let $h : X \to V_{\mathrm{hfs}}(Y)$ be an arbitrary mapping. We want to show that there exists a unique extension of h to a mapping $\hat{h} : V_{\text{hfs}}(Y) \to V_{\text{hfs}}(Y)$ that is homomorphic with respect to union "$\cup$" and (unary) set construction $\{\cdot\}$. Each $M \in V_{\text{hfs}}(Y)$ can uniquely be represented in the form $M = x_1 \cup \ldots \cup x_k \cup \{M_1\} \cup \ldots \cup \{M_l\}$ where $x_i \in X$, for $1 \leq i \leq k$, and where the M_i are the elements of M that belong to $V_{\text{hfs}}(Y)$. By induction (on nesting depth), we may assume that $\hat{h}(M_i)$ is already defined $(1 \leq i \leq l)$. Obviously $\hat{h}(M) := h(x_1) \cup \ldots \cup h(x_k) \cup \{\hat{h}(M_1)\} \cup \ldots \cup \{\hat{h}(M_l)\}$ is one and the only way of extending $\hat{h}$ in a homomorphic way to the set M of deeper nesting. For $M = x \in X$ we obtain $\hat{h}(x) = h(x)$, thus $\hat{h}$ is an extension of h. Moreover, each mapping $\hat{h}$ is in fact homomorphic with respect to union "$\cup$" and unary set construction "$\{\cdot\}$". In addition, each set $M \in V_{\text{hfs}}(Y)$ involves only finitely many different urelements (induction on the nesting depth). Thus, $V_{\text{hfs}}(Y)$, with union "$\cup$" and unary set construction "$\{\cdot\}$", is a quasi-free structure with atom set X.

Hereditarily finite non-wellfounded sets. Similarly it can be seen that the domain $V_{\text{hfnws}}(Y)$ of hereditarily finite *non-wellfounded* sets[3] over a countably infinite set of urelements Y, with union "$\cup$" and set construction "$\{\cdot\}$", is a quasi-free structure over the atom set $X = \{\{y\} \mid y \in Y\}$.

Hereditarily finite wellfounded or non-wellfounded lists. The two domains $V_{\text{hfl}}(Y)$ and $V_{\text{hfnwl}}(Y)$ of nested, hereditarily finite (1) wellfounded or (2) non-wellfounded lists over the countably infinite set of urelements Y, with concatenation "$\circ$" as binary operation and with (unary) list construction $\langle\cdot\rangle : l \mapsto \langle l \rangle$, are quasi-free structures over the atom set $X = \{\langle y \rangle \mid y \in Y\}$ of all lists with one element $y \in Y$. Formally, these domains can be described as the set of all (1) finite or (2) rational trees where the topmost node has label "$\langle\ \rangle$" (representing a list constructor of varying finite arity), nodes with successors have label "$\langle\ \rangle$", and leaves have labels $y \in Y$.

Feature structures. Let *Lab*, *Fea*, and X be mutually disjoint infinite sets of labels, features, and atoms respectively. Following [2], we define a feature tree to be a partial function $t : Fea^* \to Lab \cup X$ whose domain is prefix closed (i.e., if $pq \in \text{dom}(t)$ then $p \in \text{dom}(t)$ for all words $p, q \in Fea^*$), and in which atoms do not label interior nodes (i.e., if $t(p) = x \in X$ then there is no $f \in Fea$ with $pf \in \text{dom}(t)$). As usual, *rational* feature trees are required to have only finitely many subtrees. In addition, they must be finitely branching.

We use the set R of all rational feature trees as carrier set of a structure $\mathcal{R}^\Sigma$ whose signature contains a unary predicate L for every label $L \in Lab$, and a binary predicate f for every $f \in Fea$. The interpretation $L_\mathcal{R}$ of L in $\mathcal{R}$ is the set of all rational feature trees having root label L. The interpretation $f_\mathcal{R}$ of f consists of all pairs $(t_1, t_2) \in R \times R$ such that $t_1(f)$ is defined and t_2 is the subtree of t_1 at f. The structure $\mathcal{R}^\Sigma$ defined this way can be seen as a non-ground version of the solution domain used in [2]. We will call $\mathcal{R}^\Sigma$ the non-ground structure of rational feature trees. We will show that the set of

[3]Non-wellfounded sets, sometimes called hypersets, became prominent through [1]. They can have infinite descending membership sequences. The hereditarily finite non-wellfounded sets are those having a "finite picture", see [1] for details.

feature trees that consist of a single leaf node that is labelled by an element of X is an atom set of $\mathcal{R}^\Sigma$ w.r.t. a certain monoid $\mathcal{M}$ (see Remark 3.2.16). We identify this set in the obvious way with X.

Each mapping $h : X \to R$ has a unique extension to an endomorphism of $\mathcal{R}^\Sigma$ that acts like a substitution, replacing each leaf with label $x \in X$ by the feature tree $h(x)$. With composition, the set of these substitution-like endomorphisms yields a monoid $\mathcal{M}$. Thus, it is not difficult to see that $(\mathcal{R}^\Sigma, X)$ is quasi-free with respect to $\mathcal{M}$. However, $\mathcal{R}^\Sigma$ has endomorphisms (not belonging to $\mathcal{M}$) that modify non-leaf nodes (e.g., by introducing new feature-edges for such internal nodes). Since these modifications of non-leaf nodes are independent of the images of elements of X, the set X is not an atom set w.r.t. all endomorphisms, and thus $(\mathcal{R}^\Sigma, X)$ is not quasi-free.

Now suppose that we introduce, following [19, 104], additional arity predicates F for every finite set $F \subseteq Fea$. The interpretation $F_\mathcal{R}$ of F consists of all feature trees t where the root of t has a label $L \in Lab$ and where F is (exactly) the set of all features departing from the root of t. Let Δ be the extended signature. Then $(\mathcal{R}^\Delta, X)$ is a quasi-free structure. We shall call it the non-ground structure of rational feature trees with arity.

As can be seen from the previous examples, there is often an interesting ground variant of a given quasi-free structure. The following definition formalises this relationship.

Definition 3.2.18 Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure such that $\mathrm{SH}^{\mathfrak{A}}_\Sigma(\emptyset)$ is non-empty. Then $\mathfrak{A}^\Sigma_G := \mathrm{SH}^{\mathfrak{A}}_\Sigma(\emptyset)$ is called the *ground substructure* of $(\mathfrak{A}^\Sigma, X)$.

3.2.3 Algebraic Properties of Quasi-free Structures

Before we can turn to the combination of quasi-free structures, we must establish some useful properties of these structures.

Lemma 3.2.19 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure.*

1. $\mathfrak{A}^\Sigma = \mathrm{SH}^{\mathfrak{A}}_\Sigma(X)$ *and every mapping $X \to A$ has a* unique *extension to an endomorphism of $\mathfrak{A}^\Sigma$.*
2. *Let $X_0 \subseteq X$. Then we have $\mathrm{SH}^{\mathfrak{A}}_\Sigma(X_0) \cap X = X_0$.*
3. *For all finite sets $\{a_1, \ldots, a_n\} \subseteq A$ there exists a* unique minimal *finite subset Y of X such that $\{a_1, \ldots, a_n\} \subseteq \mathrm{SH}^{\mathfrak{A}}_\Sigma(Y)$.*

The third statement of the lemma shows that the notion "is stabilised by" behaves better than the notion "is generated by." In fact, minimal sets of generators need not be unique, as demonstrated by the next example.

Example 3.2.20 We consider the quotient term algebra $\mathcal{T}(\Sigma_F, V)/_{=_E}$, where Σ_F consists of one unary function symbol f, V is countably infinite, and $E = \{f(x) = f(y)\}$. Obviously, the carrier of $\mathcal{T}(\Sigma_F, V)/_{=_E}$ consists of the $=_E$-classes $\{x_i\}$ for $x_i \in V$ and one additional class $[f(\cdot)] := \{f(t) \mid t \in T(\Sigma_F, V)\}$. It is easy to see that for all $x_i \in V$, the element $[f(\cdot)]$ of $\mathcal{T}(\Sigma_F, V)/_{=_E}$ is generated by $\{x_i\}$. However, $[f(\cdot)]$ is not generated by $\emptyset$. Thus, there are infinitely many minimal sets of generators of $[f(\cdot)]$.

Definition 3.2.21 Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure, and let $\{a_1, \ldots, a_n\} \subseteq A$. The *stabiliser* $\mathrm{Stab}_\Sigma^{\mathfrak{A}}(a_1, \ldots, a_n)$ of $\{a_1, \ldots, a_n\}$ is the (unique) minimal finite subset Y of X such that $\{a_1, \ldots, a_n\} \subseteq \mathrm{SH}_\Sigma^{\mathfrak{A}}(Y)$.

For the case of term algebras (i.e., absolutely free algebras), the stabiliser of a term is the set of variables (i.e., generators) occurring in this term. In the more general case of arbitrary quasi-free structures, using this as an intuition will help to understand the definitions and proofs. Note, however, that the notion of a stabiliser is still well-defined (and turns out to be extremely useful) in contexts where "*the* minimal set of generators occurring in an element" is no longer unique. The next lemma is an immediate consequence of Definition 3.2.21 and of the definition of the stable hull.

Lemma 3.2.22 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure, and let Y be a subset of X. Then* $\mathrm{SH}_\Sigma^{\mathfrak{A}}(Y) = \{a \in A \mid \mathrm{Stab}_\Sigma^{\mathfrak{A}}(a) \subseteq Y\}$.

The stabilising effect of $\mathrm{Stab}_\Sigma^{\mathfrak{A}}(a)$ for a is not restricted to $End_{\mathfrak{A}}^\Sigma$. Under suitable conditions on the Σ-structure $\mathfrak{D}^\Sigma$, $\mathrm{Stab}_\Sigma^{\mathfrak{A}}(a)$ stabilises a with respect to Hom_{A-D}^Σ. Before we can formulate this in a more precise way, we must generalise Definition 3.2.8 to the quasi-free case.

Definition 3.2.23 Let $\mathfrak{A}^\Sigma, \mathfrak{D}^\Sigma$ be Σ-structures, and let $X \subseteq A$. $(\mathfrak{A}^\Sigma, X)$ is called *quasi-free for* $\mathfrak{D}^\Sigma$ if every mapping $X \to D$ has a unique extension to a homomorphism $h_{A-D} \in Hom_{A-D}^\Sigma$.

Thus, the fact that a structure $(\mathfrak{A}^\Sigma, X)$ is quasi-free is the special case where $(\mathfrak{A}^\Sigma, X)$ is quasi-free for itself.

Lemma 3.2.24 *Let $(\mathfrak{A}^\Sigma, X)$ be quasi-free, and assume that $(\mathfrak{A}^\Sigma, X)$ is quasi-free for $\mathfrak{D}^\Sigma$. Let $h_1, h_2 \in Hom_{A-D}^\Sigma$, $a \in A$ and $Y \subseteq X$.*

1. *If h_1 and h_2 coincide on* $\mathrm{Stab}_\Sigma^{\mathfrak{A}}(a)$, *then $h_1(a) = h_2(a)$.*
2. *If h_1 and h_2 coincide on Y, then h_1 and h_2 coincide on* $\mathrm{SH}_\Sigma^{\mathfrak{A}}(Y)$.

In Section 3.3.2, where we introduce a construction that combines quasi-free structures over disjoint signatures, we need to embed a given quasi-free structure into an isomorphic superstructure. Here, the usual notion of isomorphism between structures is not sufficient, however, since the atom sets must also be taken into account.

Definition 3.2.25 Let $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{B}^\Sigma, Y)$ be quasi-free. A *qf-isomorphism* between $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{B}^\Sigma, Y)$ is an isomorphism $h : \mathfrak{A}^\Sigma \to \mathfrak{B}^\Sigma$ that maps X onto Y.

The next lemma shows that qf-isomorphic structures are quasi-free for the same class of structures (in the sense introduced in Definition 3.2.23).

Lemma 3.2.26 *Let $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{B}^\Sigma, Y)$ be qf-isomorphic quasi-free structures, and let D^Σ be a Σ-structure. If $(\mathfrak{A}^\Sigma, X)$ is quasi-free for $\mathfrak{D}^\Sigma$, then also $(\mathfrak{B}^\Sigma, Y)$ is quasi-free for $\mathfrak{D}^\Sigma$. In particular, since any quasi-free structure is quasi-free for itself, $(\mathfrak{A}^\Sigma, X)$ is quasi-free for $\mathfrak{B}^\Sigma$ and $(\mathfrak{B}^\Sigma, Y)$ is quasi-free for $\mathfrak{A}^\Sigma$.*

The following two results show that one can always find qf-isomorphic substructures and superstructures of a given quasi-free structure. For free structures, showing these results is almost trivial. For quasi-free structures it requires rather long and tedious technical proofs (see [11] for details).

Lemma 3.2.27 *Let $(\mathfrak{B}^\Sigma, Y)$ be a quasi-free structure. Let Z be an infinite subset of Y, and let $\mathfrak{C}^\Sigma := \mathrm{SH}_\Sigma^{\mathfrak{B}}(Z)$. Then the following holds:*

1. *$(\mathfrak{C}^\Sigma, Z)$ is quasi-free, and $(\mathfrak{B}^\Sigma, Y)$ and $(\mathfrak{C}^\Sigma, Z)$ are qf-isomorphic.*
2. *For each $c \in C$, we have $\mathrm{Stab}_\Sigma^{\mathfrak{B}}(c) = \mathrm{Stab}_\Sigma^{\mathfrak{C}}(c)$.*
3. *For each $U \subseteq Z$, $\mathrm{SH}_\Sigma^{\mathfrak{B}}(U) = \mathrm{SH}_\Sigma^{\mathfrak{C}}(U)$.*

Theorem 3.2.28 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure. Then there exists a quasi-free superstructure $(\mathfrak{B}^\Sigma, Y)$ with the following properties:*

1. *$Y \setminus X$ is infinite.*
2. *$X \subseteq Y$, and $\mathfrak{A}^\Sigma = \mathrm{SH}_\Sigma^{\mathfrak{B}}(X)$.*
3. *$(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{B}^\Sigma, Y)$ are qf-isomorphic.*
4. *If $X \subseteq Z \subseteq Y$, and if $\mathfrak{C}^\Sigma = \mathrm{SH}_\Sigma^{\mathfrak{B}}(Z)$, then $\mathfrak{A}^\Sigma = \mathrm{SH}_\Sigma^{\mathfrak{C}}(X)$, and $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{C}^\Sigma, Z)$ are qf-isomorphic.*

3.2.4 Logical Properties of Quasi-free Structures

Using the notion of stabilisers, the validity of positive formulae in quasi-free structures can be characterised in an algebraic way. This characterisation is essential for the correctness proof (in [15]) of combining constraint solvers for quasi-free structures.

Lemma 3.2.29 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure, and let*

$$\gamma = \forall \vec{u}_1 \exists \vec{v}_1 \ldots \forall \vec{u}_k \exists \vec{v}_k \ \varphi(\vec{u}_1, \vec{v}_1, \ldots, \vec{u}_k, \vec{v}_k)$$

be a positive Σ-sentence, where φ is a positive (not necessarily quantifier-free) formula. Then the following conditions are equivalent:

1. $\mathfrak{A}^\Sigma \models \forall \vec{u}_1 \exists \vec{v}_1 \ldots \forall \vec{u}_k \exists \vec{v}_k \ \varphi(\vec{u}_1, \vec{v}_1, \ldots, \vec{u}_k, \vec{v}_k)$.
2. *There exist $\vec{x}_1 \in \vec{X}, \vec{e}_1 \in \vec{A}, \ldots, \vec{x}_k \in \vec{X}, \vec{e}_k \in \vec{A}$ such that*
   (a) $\mathfrak{A}^\Sigma \models \varphi(\vec{x}_1, \vec{e}_1, \ldots, \vec{x}_k, \vec{e}_k)$,
   (b) *all atoms in the sequences $\vec{x}_1, \ldots, \vec{x}_k$ are distinct,*
   (c) *for all $j, 1 \leq j \leq k$, the components of $\vec{x}_j$ are not contained in $\mathrm{Stab}_\Sigma^{\mathfrak{A}}(\vec{e}_1) \cup \ldots \cup \mathrm{Stab}_\Sigma^{\mathfrak{A}}(\vec{e}_{j-1})$.*

The role of the second condition of the lemma is very similar to one that *linear constant restrictions* played in work on combining unification algorithms. This notion was introduced in [5]. We will present and discuss it in the chapter on optimisation techniques.

3.3 Combination of Quasi-free Structures

This section is concerned with the problem of how to combine two quasi-free structures over disjoint signatures into a new structure over the union of both signatures. First, we will introduce an algebraic framework for combining structures, which is not restricted to quasi-free structures or disjoint signatures. This framework tries to formalise our intuition of what to expect from a canonical combination of two structures. We go on to describe an explicit construction for combining two quasi-free structures over disjoint signatures, and show that the result of this construction coincides with what our abstract framework proposes as canonical combined structure.

3.3.1 Combination of Structures

Let $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ be two structures. What conditions should a $(\Sigma_1 \cup \Sigma_2)$-structure $\mathfrak{C}^{\Sigma_1 \cup \Sigma_2}$ satisfy to be called a "canonical combination" of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$? The central notion of this subsection will be obtained after three steps, each introducing a restriction that is motivated by the example of the combination of free algebras, i.e., term algebras modulo equational theories. The structures $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ will be called the *components* in the sequel.

> **Restriction 1:** *Homomorphisms that embed the components into the combined structure must exist. If the components share a common substructure, then the embedding homomorphisms must agree on this substructure.*

In fact, a minimal requirement seems to be that both structures must in some sense be embedded in their combination. It would, however, be too restrictive to demand that the components are substructures of the combined structure. For the case of consistent equational theories E_1, E_2 over disjoint signatures Σ_1, Σ_2, there exist 1–1-embeddings of $\mathcal{T}(\Sigma_1, V)/_{=E_1}$ and $\mathcal{T}(\Sigma_2, V)/_{=E_2}$ into $\mathcal{T}(\Sigma_1 \cup \Sigma_2, V)/_{=E_1 \cup E_2}$. For non-disjoint signatures, however, these "embeddings" need no longer be 1–1. Note that even for disjoint signatures Σ_1 and Σ_2 there is a common part, namely the trivial structure represented by the set V of variables. A reasonable requirement is that elements of the common part are mapped to the same element of the combined structure by the homomorphic embeddings. To be as general as possible, we do not assume that the "common part" is really a substructure of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$. Instead, we assume that it is just homomorphically embedded in both structures. These considerations motivate the following formalisation of Restriction 1.

Definition 3.3.1 Let Σ_1 and Σ_2 be signatures, and let $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$. A triple $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ with given homomorphic embeddings

$$h^\Gamma_{A-B_1} : \mathfrak{A}^\Gamma \to \mathfrak{B}_1^{\Sigma_1} \quad \text{and} \quad h^\Gamma_{A-B_2} : \mathfrak{A}^\Gamma \to \mathfrak{B}_2^{\Sigma_2}$$

is called an *amalgamation base*. The structure $\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}$ *closes* the amalgamation base $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ iff there are homomorphisms

$$h^{\Sigma_1}_{B_1-D} : \mathfrak{B}_1^{\Sigma_1} \to \mathfrak{D}^{\Sigma_1} \quad \text{and} \quad h^{\Sigma_2}_{B_2-D} : \mathfrak{B}_2^{\Sigma_2} \to \mathfrak{D}^{\Sigma_2}$$

such that $h^{\Sigma_1}_{B_1-D} \circ h^\Gamma_{A-B_1} = h^{\Sigma_2}_{B_2-D} \circ h^\Gamma_{A-B_2}$. We call $(\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}, h^{\Sigma_1}_{B_1-D}, h^{\Sigma_2}_{B_2-D})$ an *amalgamated product* of $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$.

If the embedding homomorphisms are irrelevant or clear from the context, we will also call the structure $\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}$ alone an amalgamated product of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ over $\mathfrak{A}^\Gamma$. For a given amalgamation base, there usually exist various structures that can be used to close this base. Which one should be seen as a canonical closure? Motivated by the example of free structures, where the canonical combined structure is again free, we are interested in "most general" amalgamated products.

> **Restriction 2:** *We are interested in structures closing the amalgamation base that are as general as possible.*

In principle, we consider a structure $\mathfrak{C}$ to be more general than a structure $\mathfrak{D}$ iff there is a homomorphism of $\mathfrak{C}$ into $\mathfrak{D}$. Thus, a possible formalisation of Restriction 2 seems to be to ask for an amalgamated product

$$(\mathfrak{C}^{\Sigma_1 \cup \Sigma_2}, h^{\Sigma_1}_{B_1-C}, h^{\Sigma_2}_{B_2-C})$$

such that for each amalgamated product $(\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}, h^{\Sigma_1}_{B_1-D}, h^{\Sigma_2}_{B_2-D})$ of the amalgamation base there exists a unique $(\Sigma_1 \cup \Sigma_2)$-homomorphism h_{C-D} such that

$$\begin{tikzcd}
& \mathfrak{B}_1^{\Sigma_1} \arrow[dr, "h_{B_1-C}"] \arrow[drr, "h_{B_1-D}"] & & \\
\mathfrak{A}^{\Gamma} \arrow[ur, "h_{A-B_1}"] \arrow[dr, "h_{A-B_2}"'] & & \mathfrak{C}^{\Sigma_1 \cup \Sigma_2} \arrow[r, "h_{C-D}"] & \mathfrak{D}^{\Sigma_1 \cup \Sigma_2} \\
& \mathfrak{B}_2^{\Sigma_2} \arrow[ur, "h_{B_2-C}"] \arrow[urr, "h_{B_2-D}"'] & &
\end{tikzcd}$$

$h_{B_i-D} = h_{C-D} \circ h_{B_i-C}$, for $i = 1, 2$. This situation is illustrated in the following figure.

It turns out, however, that requiring a most general element among *all* possible amalgamated products is too strong. Informally, the reason is that not all amalgamated products of a given amalgamation base share "relevant" structural properties with the component structures of the base. To be more precise, we consider the example of free algebras $\mathfrak{B}_1^{\Sigma_1} := \mathcal{T}(\Sigma_1, V)/_{=E_1}$ and $\mathfrak{B}_2^{\Sigma_2} := \mathcal{T}(\Sigma_2, V)/_{=E_2}$, with common "substructure" $\mathfrak{A}^{\Gamma} := \mathcal{T}(\Sigma_1 \cap \Sigma_2, V)$. The canonical combined algebra is the free algebra $\mathcal{T}(\Sigma_1 \cup \Sigma_2, V)/_{=E_1 \cup E_2}$, which is in fact most general (in the sense introduced above) among all amalgamated products that satisfy $E_1 \cup E_2$, i.e., all elements of $\mathcal{V}(E_1 \cup E_2)$. An arbitrary product $\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}$ of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ may, however, invalidate some axioms of $E_1 \cup E_2$. In this case, it may not be possible to find an appropriate homomorphism from $\mathcal{T}(\Sigma_1 \cup \Sigma_2, V)/_{=E_1 \cup E_2}$ to $\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}$ (see [11] for an example). For this reason, we allow for the possibility of restricting the attention to a certain subclass of all amalgamated products.

> **Restriction 3:** *Only admissible combinations of the two components are considered. The class of admissible structures should share relevant structural properties with these components.*

For the case of free algebras, the obvious candidate for the class of admissible structures is the the variety defined by the union of the component theories, i.e., $Adm(\mathcal{T}(\Sigma_1, V)/_{=E_1}, \mathcal{T}(\Sigma_2, V)/_{=E_2}) = \mathcal{V}(E_1 \cup E_2)$. An appropriate class of admissible structures for the quasi-free case will be obtained as a generalisation of this. In the remainder of this subsection, however, we make no assumption on the specific form of the class of admissible structures. We just assume that such a class is given. An amalgamated product is called admissible iff it belongs to the class of admissible structures.

Definition 3.3.2 Let $(\mathfrak{A}^{\Gamma}, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ be an amalgamation base, and assume that a class $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ of admissible $(\Sigma_1 \cup \Sigma_2)$-structures is fixed. The admissible amalgamated product $(\mathfrak{C}^{\Sigma_1 \cup \Sigma_2}, h_{B_1-C}^{\Sigma_1}, h_{B_2-C}^{\Sigma_2})$ of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ over $\mathfrak{A}^{\Gamma}$ is called a *free amalgamated product with respect to* $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ iff for every admissible amalgamated product $(\mathfrak{D}^{\Sigma_1 \cup \Sigma_2}, h_{B_1-D}^{\Sigma_1}, h_{B_2-D}^{\Sigma_2})$ of $\mathfrak{B}_1^{\Sigma_1}$ and

$\mathfrak{B}_2^{\Sigma_2}$ over $\mathfrak{A}^\Gamma$ there exists a *unique* homomorphism $h_{C-D}^{\Sigma_1\cup\Sigma_2} : \mathfrak{C}^{\Sigma_1\cup\Sigma_2} \to \mathfrak{D}^{\Sigma_1\cup\Sigma_2}$ such that

$$h_{B_1-D}^{\Sigma_1} = h_{C-D}^{\Sigma_1\cup\Sigma_2} \circ h_{B1-C}^{\Sigma_1} \quad \text{and} \quad h_{B_2-D}^{\Sigma_2} = h_{C-D}^{\Sigma_1\cup\Sigma_2} \circ h_{B_2-C}^{\Sigma_2}.$$

Free amalgamated products need not exist, but if they exist they are unique up to isomorphism.

Theorem 3.3.3 *Let $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ be an amalgamation base with fixed homomorphic embeddings $h_{A-B_1}^\Gamma : \mathfrak{A}^\Gamma \to \mathfrak{B}_1^{\Sigma_1}$ and $h_{A-B_2}^\Gamma : \mathfrak{A}^\Gamma \to \mathfrak{B}_2^{\Sigma_2}$. The free amalgamated product of $\mathfrak{B}_1^{\Sigma_1}$ and $\mathfrak{B}_2^{\Sigma_2}$ over $\mathfrak{A}^\Gamma$ with respect to a given class $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2})$ is unique up to $(\Sigma_1 \cup \Sigma_2)$-isomorphism.*

The theorem justifies to speak about *the* free amalgamated product of two structures (provided that the embedding homomorphisms and the class of admissible structures are fixed). In this situation, we will sometimes denote the free amalgamated product of $\mathfrak{B}_1$ and $\mathfrak{B}_2$ by $\mathfrak{B}_1 \otimes \mathfrak{B}_2$. The product operation is obviously commutative, if the definition of the class of admissible structures satisfies $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}) = Adm(\mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_1^{\Sigma_1})$. In order to obtain associativity as well, we need some additional conditions on the class of admissible structures.

Before formulating these restrictions, we extend the definition of an amalgamation base and of the free amalgamated product to the case of three structures.[4] Let $\Gamma \subseteq \Sigma_1 \cap \Sigma_2 \cap \Sigma_3$. A quadruple $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3})$ with given homomorphic embeddings

$$h_{A-B_i}^\Gamma : \mathfrak{A}^\Gamma \to \mathfrak{B}_i^{\Sigma_i} \quad (i = 1, 2, 3)$$

is called a *simultaneous amalgamation base*. The structure $\mathfrak{D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3}$ *closes* the simultaneous amalgamation base $(\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3})$ iff, for $i = 1, 2, 3$, there are homomorphisms $h_{B_i-D}^{\Sigma_i} : \mathfrak{B}_i^{\Sigma_i} \to \mathfrak{D}^{\Sigma_i}$ such that

$$h_{B_1-D}^{\Sigma_1} \circ h_{A-B_1}^\Gamma = h_{B_2-D}^{\Sigma_2} \circ h_{A-B_2}^\Gamma = h_{B_3-D}^{\Sigma_3} \circ h_{A-B_3}^\Gamma.$$

In this case, $(\mathfrak{D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3}, h_{B_1-D}^{\Sigma_1}, h_{B_2-D}^{\Sigma_2}, h_{B_3-D}^{\Sigma_3})$ is a *simultaneous amalgamated product* of $\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}$ over $\mathfrak{A}^\Gamma$.

Now, assume that a class of admissible structures $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3})$ is fixed. The admissible simultaneous amalgamated product

$$(\mathfrak{C}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3}, h_{B_1-C}^{\Sigma_1}, h_{B_2-C}^{\Sigma_2}, h_{B_3-C}^{\Sigma_3})$$

of $\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}$ over $\mathfrak{A}^\Gamma$ is called a *free simultaneous amalgamated product with respect to* $Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3})$ iff for every admissible simultaneous amalgamated product $(\mathfrak{D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3}, h_{B_1-D}^{\Sigma_1}, h_{B_2-D}^{\Sigma_2}, h_{B_3-D}^{\Sigma_3})$ there exists a *unique* homomorphism

$$f_{C-D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3} : \mathfrak{C}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3} \to \mathfrak{D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3}$$

such that $g_{B_i-D}^{\Sigma_i} = f_{C-D}^{\Sigma_1\cup\Sigma_2\cup\Sigma_3} \circ h_{B_i-C}^{\Sigma_i}$, for $i = 1, 2, 3$. As for the binary free amalgamated product, one can show that the free simultaneous amalgamated product is unique up to isomorphism, provided that it exists.

[4] The extension to an arbitrary number $n \geq 2$ of structures should then be obvious.

Theorem 3.3.4 (Associativity of free amalgamation)
Let $\Gamma \subseteq \Sigma_1 \cap \Sigma_2 \cap \Sigma_3$, and let $\mathfrak{A}^\Gamma, \mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}$ be structures with fixed homomorphic embeddings $h^\Gamma_{A-B_1} : \mathfrak{A}^\Gamma \to \mathfrak{B}_1^{\Sigma_1}$, $h^\Gamma_{A-B_2} : \mathfrak{A}^\Gamma \to \mathfrak{B}_2^{\Sigma_2}$, and $h^\Gamma_{A-B_3} : \mathfrak{A}^\Gamma \to \mathfrak{B}_3^{\Sigma_3}$. Assume that the free amalgamated products $\mathfrak{B}_2^{\Sigma_2} \otimes \mathfrak{B}_3^{\Sigma_3}$, $\mathfrak{B}_1^{\Sigma_1} \otimes (\mathfrak{B}_2^{\Sigma_2} \otimes \mathfrak{B}_3^{\Sigma_3})$, $\mathfrak{B}_1^{\Sigma_1} \otimes B_2^{\Sigma_2}$, and $(\mathfrak{B}_1^{\Sigma_1} \otimes B_2^{\Sigma_2}) \otimes \mathfrak{B}_3^{\Sigma_3}$ exist, and that the classes of admissible structures satisfy

$$\begin{array}{rcl}
\mathfrak{B}_1^{\Sigma_1} \otimes (\mathfrak{B}_2^{\Sigma_2} \otimes \mathfrak{B}_3^{\Sigma_3}) & \in & Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}), \\
(\mathfrak{B}_1^{\Sigma_1} \otimes \mathfrak{B}_2^{\Sigma_2}) \otimes \mathfrak{B}_3^{\Sigma_3} & \in & Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}), \text{ and} \\
Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}) & \subseteq & Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2}) \cap Adm(\mathfrak{B}_1^{\Sigma_1} \otimes \mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}) \cap \\
 & & Adm(\mathfrak{B}_2^{\Sigma_2}, \mathfrak{B}_3^{\Sigma_3}) \cap Adm(\mathfrak{B}_1^{\Sigma_1}, \mathfrak{B}_2^{\Sigma_2} \otimes \mathfrak{B}_3^{\Sigma_3}).
\end{array}$$

Then we have $(\mathfrak{B}_1^{\Sigma_1} \otimes \mathfrak{B}_2^{\Sigma_2}) \otimes \mathfrak{B}_3^{\Sigma_3} \simeq \mathfrak{B}_1^{\Sigma_1} \otimes (\mathfrak{B}_2^{\Sigma_2} \otimes \mathfrak{B}_3^{\Sigma_3})$, and this structure is the free simultaneous amalgamated product of $\mathfrak{B}_1^{\Sigma_1}$, $B_2^{\Sigma_2}$, and $\mathfrak{B}_3^{\Sigma_3}$ over $\mathfrak{A}^\Gamma$.

The proof of this theorem, which can be found in [11], can be given on a rather abstract level (manipulation of arrows, i.e., homomorphisms). Note, however, that proving in a particular situation that the prerequisites of the theorem are satisfied is usually not possible on this abstract external level; it may require deep knowledge about the internal structure of the involved structures.

Notions of "amalgamated product," similar to the one given above, can be found in universal algebra, model theory, and in category theory (see, e.g., [26, 41, 73]). There are, however, certain differences between our situation and the typical situations in which amalgamation occurs in other areas. In algebra or model theory, amalgamation has been introduced for *particular classes of algebraic structures* such as groups, fields, skew fields etc. Amalgamation is studied for such a fixed class of structures over the same signature, and it is assumed that these structures all satisfy the same set of axioms (e.g., those for groups, fields, skew fields, etc.). In our case, algebras over different signatures are amalgamated, and these algebras satisfy different types of axioms (or are not defined by axioms at all).

3.3.2 An Amalgamation Construction for Quasi-free Structures

We describe an explicit construction for closing any amalgamation base where the two components are quasi-free structures over disjoint signatures. In Section 3.3.3 we will show that the constructed amalgamated product is in fact the free amalgamated product. Having such an explicit construction rather than just an abstract algebraic characterisation of the free amalgamated product is important in the correctness proof of our method for combining constraint solvers. The description of the construction given below is considerably different from the one presented in [8, 10]. The main advantage of this new description is that it allows for shorter and simpler proofs.

Let $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ be quasi-free structures over disjoint signatures Σ_1 and Σ_2 such that $A_1 \cap A_2 = X$. We consider the amalgamation base

$(X, \mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2})$, where the common part is just the set of atoms X. Thus, for $i = 1, 2$, the embedding "homomorphisms" $h_{X-A_i} : X \to A_i^{\Sigma_i}$ are given by id_X. In order to close this amalgamation base, we first embed each component $(\mathfrak{A}_i^{\Sigma_i}, X)$ into an isomorphic superstructure $(\mathfrak{B}_i^{\Sigma_i}, Y_i)$ satisfying Conditions 1–4 of Theorem 3.2.28 ($i = 1, 2$). In addition, we assume without loss of generality that $B_1 \cap B_2 = X$. Our goal is to construct (for $i = 1, 2$) a Σ_i-structure $\mathfrak{C}_i^{\Sigma_i}$, which is a superstructure of $\mathfrak{A}_i^{\Sigma_i}$ and a substructure of $\mathfrak{B}_i^{\Sigma_i}$. The construction will provide us with a bijection between C_2 and C_1 satisfying certain properties. This bijection can be used to carry the Σ_2-structure of $\mathfrak{C}_2^{\Sigma_2}$ over to C_1. The $(\Sigma_1 \cup \Sigma_2)$-structure obtained in this way is the result of the construction. The properties of the bijection will guarantee that this result is in fact the free amalgamated product of the component structures. For defining the required bijection, the notion of a fibre will be important.

Definition 3.3.5 *Fibres* are either 1-fibres or 2-fibres. A *1-fibre* is of the form $F = \{x\}$ for $x \in X$, and a *2-fibre* is of the form $F = \{y, b\}$ where $y \in Y_i \setminus X$ and $b \in B_j \setminus Y_j$ for $\{i, j\} = \{1, 2\}$. For a fibre F and $i = 1, 2$, we define $F(i) := F \cap B_i$. The index of a 2-fibre F is j iff $F(j)$ is the non-atom element of F.

The fibring construction

Let $b_1, b_2, b_3, \ldots$ be an enumeration of $B_{1,2} := B_1 \cup B_2$. Using this enumeration, we construct an ascending tower of sets $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$ where each $\mathcal{F}_i$ is a set of mutually disjoint fibres. In addition, each set $\mathcal{F}_i$ contains only finitely many 2-fibres. We start with $\mathcal{F}_0 := \{\{x\} \mid x \in X\}$, i.e., $\mathcal{F}_0$ is the set of all 1-fibres. Now, assume that $\mathcal{F}_k$ has already been defined, and that all fibres of $\mathcal{F}_k$ are mutually disjoint. When defining $\mathcal{F}_{k+1}$, we distinguish two situations.

Case 1: If there exists an element b of $B_{1,2}$, say in B_i, such that

1. each element of the stabiliser $\mathrm{Stab}_{\Sigma_i}^{\mathfrak{B}_i}(b)$ belongs to a fibre $F \in \mathcal{F}_k$, but
2. b itself does not belong to a fibre $F \in \mathcal{F}_k$,

then we proceed as follows: Let b_{min} be the first element of $B_{1,2}$ (in the enumeration $b_1, b_2, b_3, \ldots$) satisfying the two Properties 1 and 2, and let i be such that $b_{min} \in B_i$. For the other index $j \neq i$, we select an atom $z \in Y_j$ that does not belong to any fibre of $\mathcal{F}_k$. Such an atom exists since $\mathfrak{B}_j^{\Sigma_1}$ satisfies Condition 1 of Theorem 3.2.28, and F_k is assumed to contain only finitely many 2-fibres. We define $F_{k+1} := \{b_{min}, z\}$, and $\mathcal{F}_{k+1} := \mathcal{F}_k \cup \{F_{k+1}\}$. Note that F_{k+1} is indeed a 2-fibre since b_{min} cannot be an atom. In fact, it is easy to see that any atom x has the singleton set $\{x\}$ as its stabiliser. Thus, an atom cannot satisfy the Conditions 1 and 2 simultaneously.

Case 2: Otherwise, we define $\mathcal{F}_{k+1} := \mathcal{F}_k$.

By definition, $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$, and at each level k, all fibres of $\mathcal{F}_k$ are mutually disjoint.

Let $\mathcal{F} := \bigcup_{k \geq 0} \mathcal{F}_k$. Then $C_{1,2} := \bigcup_{F \in \mathcal{F}} F$ is a subset of $B_{1,2}$. Let $C_i := C_{1,2} \cap B_i$, and $Z_i := C_{1,2} \cap Y_i$ $(i = 1, 2)$. We say that an element of $B_{1,2}$ is fibred iff it belongs to a fibre of $\mathcal{F}$. For these elements we define a height.

Definition 3.3.6 For $a \in C_{1,2}$ define the *height* of a by $\mathit{height}(a) := k$ iff a is an element of the fibre F_k.

So, the height of each $x \in X$ is 0; and the height of each non-atomic element with non-empty stabiliser is larger then the height of each element in its stabiliser.

The definition of the amalgamated structure

The sets C_1 and C_2 are indeed stable hulls.

Lemma 3.3.7 *$C_i = \mathrm{SH}_{\Sigma_i}^{\mathfrak{B}_i}(Z_i)$, and thus $\mathfrak{C}_i^{\Sigma_i}$ is a Σ_i-substructure of $\mathfrak{B}_i^{\Sigma_i}$.*

Now, we define appropriate bijections between C_1 and C_2. Each element $c \in C_{1,2}$ belongs to a unique fibre F_c of $\mathcal{F}$. We define the bijections $h_{i,j} : C_i \to C_j$ by mapping each $c \in C_i$ to $F_c(j)$, the unique element of F_c belonging to C_j $(\{i, j\} = \{1, 2\})$. Obviously this implies $h_{i,j} = {h_{j,i}}^{-1}$. Note that any element x of X belongs to a 1-fibre, and thus

$$h_{i,j}(x) = x \text{ for all } x \in X. \tag{3.1}$$

The bijections $h_{1,2}$ and $h_{2,1}$ are now used to carry the Σ_2-structure of $\mathfrak{C}_2^{\Sigma_2}$ to C_1: Let f be an n-ary function symbol of Σ_2, let p be an n-ary predicate symbol of Σ_2, and let $a_1, \ldots, a_n \in C_1$. We define

$$\begin{array}{rcl} f_{\mathfrak{C}_1}(a_1, \ldots, a_n) & := & h_{2,1}(f_{\mathfrak{C}_2}(h_{1,2}(a_1), \ldots, h_{1,2}(a_n))) \\ p_{\mathfrak{C}_1}[a_1, \ldots, a_n] & :\Longleftrightarrow & p_{\mathfrak{C}_2}[h_{1,2}(a_1), \ldots, h_{1,2}(a_n)]. \end{array}$$

In the same way, we impose the Σ_1-structure of $\mathfrak{C}^{\Sigma_1}$ on C_2. Thus, both $\mathfrak{C}_1$ and $\mathfrak{C}_2$ can be seen as $(\Sigma_1 \cup \Sigma_2)$-structures. Let $\Sigma := \Sigma_1 \cup \Sigma_2$. By construction, the mappings

$$h_{1,2} \text{ and } h_{2,1} \text{ are inverse } \Sigma\text{-isomorphisms between } \mathfrak{C}_1^{\Sigma} \text{ and } \mathfrak{C}_2^{\Sigma}. \tag{3.2}$$

For this reason, it is irrelevant which of these two structures is taken as result of the construction. In the following, we use $\mathfrak{C}_1^{\Sigma}$ as the amalgamated structure obtained by the construction, and we will sometimes denote this structure by $\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$.

Properties of the amalgamation construction

Before we can show that the construction really yields the free amalgamated product, we must state some useful properties:

$$(\mathfrak{C}_i^{\Sigma_i}, Z_i) \text{ and } (\mathfrak{A}_i^{\Sigma_i}, X) \text{ are qf-isomorphic (for } i = 1, 2). \tag{3.3}$$

(3.3) follows from the fact that (for $i = 1, 2$) $\mathfrak{B}_i^{\Sigma_i}$ satisfies Condition 4 of Theorem 3.2.28. In addition, by Lemma 3.2.27 we have

$$\forall d \in \mathfrak{C}_i : \mathrm{Stab}_{\Sigma_i}^{\mathfrak{C}_i}(d) = \mathrm{Stab}_{\Sigma_i}^{\mathfrak{B}_i}(d) \text{ and } \forall U \subseteq Z_i : \mathrm{SH}_{\Sigma_i}^{\mathfrak{C}_i}(U) = \mathrm{SH}_{\Sigma_i}^{\mathfrak{B}_i}(U) \quad (3.4)$$

For $i = 1, 2$, each set of fibres $\mathcal{F}_k$ determines a set $\mathcal{F}_k^i := \{F(i) \mid F \in \mathcal{F}_k\} \subseteq C_i$. Now, (3.4) and the definition of the fibring construction imply:

$$\text{If } c \in C_i \setminus Z_i \text{ is in } \mathcal{F}_{k+1}^i, \text{ then } \mathrm{Stab}_{\Sigma_i}^{\mathfrak{C}_i}(c) \subseteq \mathcal{F}_k^i \text{ (for } i = 1, 2). \quad (3.5)$$

In order to show that $\mathfrak{C}_1^{\Sigma}$ closes the amalgamation base $(X, \mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2})$, we define

$$h_{A_1 - C_1} := \mathrm{id}_{A_1} \text{ and } h_{A_2 - C_1} := h_{2,1}|_{A_2}. \quad (3.6)$$

By definition of $h_{A_i - C_1}$ and (3.1) we know that

$$h_{A_i - C_1}|_X = \mathrm{id}_X \text{ (for } i = 1, 2). \quad (3.7)$$

Thus, $h_{A_1 - C_1} \circ h_{X - A_1} = \mathrm{id}_X = h_{A_2 - C_1} \circ h_{X - A_2}$, which shows:

Lemma 3.3.8 *The amalgamated structure $\mathfrak{C}_1^{\Sigma}$ obtained by the construction is an amalgamated product of $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$.*

Definition 3.3.9 The enumeration $b_1, b_2, b_3, \ldots$ defines a strict linear ordering $\prec_X$ on X. In addition, a strict linear ordering $\prec_i$ on the complements $C_i \setminus X$ is given by the order in which the elements of $C_i \setminus X$ are fibred: We define $c \prec_i d$ iff, for some k, $c \in \mathcal{F}_k^i$ and $d \notin \mathcal{F}_k^i$. With $<_i$ we denote the unique strict linear ordering on C_i that extends both $\prec_X$ and $\prec_i$, and makes each element of X smaller than each element of $C_i \setminus X$ $(i = 1, 2)$.

As an easy consequence of this definition, we obtain

$$\forall c, d \in C_i\colon c <_i d \quad \text{iff} \quad h_{i,j}(c) <_j h_{i,j}(d) \quad (\{i, j\} = \{1, 2\}), \quad (3.8)$$

$$\forall c, d \in C_i\colon c <_i d \quad \text{implies} \quad d \notin \mathrm{Stab}_{\Sigma_i}^{\mathfrak{C}_i}(c) \quad (i \in \{1, 2\}). \quad (3.9)$$

Note that (3.8) is trivial, and that (3.9) follows from (3.5).

3.3.3 The Free Amalgamated Product of Quasi-free Structures

In this subsection, we will show that the amalgamation construction presented above really yields the free amalgamated product of the quasi-free component structures. In the sequel, $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ denote quasi-free structures, which are used as the input components of the amalgamation construction. We shall also refer to other entities introduced in the construction, such as $\mathfrak{C}_i^{\Sigma}, \mathfrak{B}_i^{\Sigma_i}, h_{i,j}, \mathcal{F}_k$, etc. First, we must fix the class of admissible structures with respect to which the free product is to be built.

Definition 3.3.10 Let $(X, \mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2})$ be an amalgamation base, where both $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ are quasi-free structures over disjoint signatures. Then we choose

$$Adm(\mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2}) := \{\mathfrak{D}^{\Sigma_1 \cup \Sigma_2} \mid (\mathfrak{A}_i^{\Sigma_i}, X) \text{ is quasi-free for } \mathfrak{D}^{\Sigma_i}, \text{ for } i = 1, 2\}$$

as the class of admissible structures.

Theorem 3.3.11 *$\mathfrak{C}_1^{\Sigma} = \mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ is the free amalgamated product of the quasi-free structures $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$ with respect to the class $Adm(\mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2})$ of admissible structures defined above.*

3.3.4 Multiple and Iterated Amalgamation

The explicit amalgamation construction introduced above can easily be generalised to a construction that combines an arbitrary number $n \geq 2$ of quasi-free structures over disjoint signatures.[5] The – here omitted – proof for the theorem in the above subsection can also be generalised to show that the extended construction yields the n-fold simultaneous free amalgamated product, provided that the following obvious generalisation of the class of admissible structures is used:

$$\begin{aligned} &Adm(\mathfrak{A}_1^{\Sigma_1}, \ldots, \mathfrak{A}_n^{\Sigma_n}) = \\ &\qquad \{\mathfrak{D}^{\Sigma_1 \cup \ldots \cup \Sigma_n} \mid \mathfrak{A}_i^{\Sigma_i} \text{ is quasi-free for } \mathfrak{D}^{\Sigma_i}, \text{ for } 1 \leq i \leq n\}. \end{aligned} \tag{3.10}$$

In this subsection, we show that it is not really necessary to introduce the explicit amalgamation construction for the case $n > 2$ since the free amalgamated product can also be obtained by iterated application of the construction to two structures. Obviously, iterated application is only possible if the structure obtained by the construction is again quasi-free. The following proposition shows that this prerequisite is satisfied.

Proposition 3.3.12 *The free amalgamated product of two quasi-free structures with common atom set X is a quasi-free structure with atom set X.*

Corollary 3.3.13 *$(\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}, X)$ is a quasi-free structure that is quasi-free for each $\mathfrak{D}^{\Sigma} \in Adm(\mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2})$.*

Obviously, the set of admissible structures, as introduced in Definition 3.3.10, satisfies $Adm(\mathfrak{A}_1^{\Sigma_1}, \mathfrak{A}_2^{\Sigma_2}) = Adm(\mathfrak{A}_2^{\Sigma_2}, \mathfrak{A}_1^{\Sigma_1})$. Thus, free amalgamation of quasi-free structures is commutative. Since the amalgamation construction can be iterated, the question arises whether the construction is associative as well. This question is answered to the affirmative by showing that the assumptions of Theorem 3.3.4 are satisfied. Thereby, the theorem also shows that simultaneous free Amalgamation and iterated free Amalgamation yield the same result.

[5] It is even possible to amalgamate a countably infinite number of quasi-free structures in this way.

Theorem 3.3.14 *Free amalgamation of quasi-free structures with disjoint signatures over the same atom set is associative, and free simultaneous amalgamation coincides with iterated free amalgamation.*

3.4 Combining Constraint Solvers for Quasi-free Structures

Let $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ be quasi-free structures over disjoint signatures Σ_1 and Σ_2, and let $\mathfrak{C}_1^{\Sigma} = \mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ denote their free amalgamated product, as constructed in the previous section, where $\Sigma = \Sigma_1 \cup \Sigma_2$. This section is devoted to the presentation of the following combination result for constraint solvers over quasi-free structures.

Theorem 3.4.1 *The positive theory of $\mathfrak{C}_1^{\Sigma} = \mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ is decidable, provided that the positive theories of the quasi-free structures $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$ are decidable.*

First, we show how constraint solvers for the positive theories of $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$ can be combined to a constraint solver for the *existential* positive theory of $\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$. In a second subsection, it is shown that this result can be lifted to the full positive theory of $\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$.

3.4.1 The Existential Positive Case

In this subsection, we present a restricted version of Theorem 3.4.1.

Theorem 3.4.2 *The existential positive theory of $\mathfrak{C}_1^{\Sigma} = \mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ is decidable, provided that the positive theories of the quasi-free structures $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$ are decidable.*

The same theorem can be proved for the simultaneous free amalgamated product of $n \geq 2$ quasi-free components over disjoint signatures. To keep things simpler, we restrict our attention to the case $n = 2$.

The decomposition algorithm described below decomposes an existential positive Σ-sentence φ_0 into a finite set of pairs (α, β), where α is a positive Σ_1-sentence and β is a positive Σ_2-sentence. This algorithm coincides with the one described in [8], where it has been used in the restricted context of combination problems for free structures. Steps similar to Step 1, 3, and the labelling in Step 4 are present in most methods for combining unification algorithms. Nelson & Oppen's combination method for universal theories [77] explicitly uses Step 1, and implicitly, Step 3 is also present.

Before we can describe the algorithm, we must introduce some notation. In the following, V denotes an infinite set of variables used by the first-order languages under consideration. Let t be a Σ-term. This term is called *pure* iff it is either a Σ_1-term or a Σ_2-term. An equation is pure iff it is an equation between pure terms of the same signature. A relational formula $p[s_1, \ldots, s_m]$ is pure iff $s_1, \ldots, s_m$ are pure terms of the signature of p. Now assume that t is a non-pure term whose topmost function symbol is in Σ_1. A subterm s of t is called *alien subterm* of t iff its topmost function symbol belongs to Σ_2 and every proper superterm of s in t has its top symbol in Σ_1. Alien subterms of terms with top symbol in Σ_2 are defined analogously. For a relational formula $p[s_1, \ldots, s_m]$, alien subterms are defined as follows: if s_i has a top symbol whose signature is different from the signature of p then s_i itself is an alien subterm; otherwise, any alien subterm of s_i is an alien subterm of $p[s_1, \ldots, s_m]$.

The decomposition algorithm

Let φ_0 be an existential positive Σ-sentence. Without loss of generality, we may assume that φ_0 has the form $\exists \vec{u}_0\ \gamma_0$, where γ_0 is a conjunction of atomic formulae. Indeed, since existential quantifiers distribute over disjunction, a sentence $\exists \vec{u}_0\ (\gamma_1 \vee \gamma_2)$ is valid iff $\exists \vec{u}_0\ \gamma_1$ or $\exists \vec{u}_0\ \gamma_2$ is valid.

Step 1: Transform non-pure atomic formulae.

(1) Equations $s = t$ of γ_0 where s and t have topmost function symbols belonging to different signatures are replaced by (the conjunction of) two new equations $u = s, u = t$, where u is a new variable. The quantifier prefix is extended by adding an existential quantification for u.

(2) As a result, we may assign a unique label Σ_1 or Σ_2 to each atomic formula that is not an equation between variables. The label of an equation $s = t$ is the signature of the topmost function symbols of s and/or t. The label of a relational formula $p[s_1, \ldots, s_m]$ is the signature of p.

(3) Now alien subterms occurring in atomic formulae are successively replaced by new variables. For example, assume that $s = t$ is an equation in the current formula, and that s contains the alien subterm s_1. Let u be a variable not occurring in the current formula, and let s' be the term obtained from s by replacing s_1 by u. Then the original equation is replaced by (the conjunction of) the two equations $s' = t$ and $u = s_1$. The quantifier prefix is extended by adding an existential quantification for u. The equation $s' = t$ keeps the label of $s = t$, and the label of $u = s_1$ is the signature of the top symbol of s_1. Relational atomic formulae with alien subterms are treated analogously. This process is iterated until all atomic formulae occurring in the conjunctive matrix are pure. It is easy to see that this is achieved after finitely many iterations.

Step 2: Remove atomic formulae without label.

Equations between variables occurring in the conjunctive matrix are removed as follows: If $u = v$ is such an equation then one removes $\exists u$ from the quantifier prefix and $u = v$ from the matrix. In addition, every occurrence of u in the remaining matrix is replaced by v. This step is iterated until the matrix contains no equations between variables.

Let φ_1 be the new sentence obtained this way. The matrix of φ_1 can be written as a conjunction $\gamma_{1,\Sigma_1} \wedge \gamma_{1,\Sigma_2}$, where γ_{1,Σ_1} is a conjunction of all atomic formulae from φ_1 with label Σ_1, and γ_{1,Σ_2} is a conjunction of all atomic formulae from φ_1 with label Σ_2. There are three different types of variables occurring in φ_1: *shared variables* occur both in γ_{1,Σ_1} and in γ_{1,Σ_2}; Σ_1-*variables* occur only in γ_{1,Σ_1}; and Σ_2-*variables* occur only in γ_{1,Σ_2}. Let $\vec{u}_{1,\Sigma_1}$ be the tuple of all Σ_1-variables, $\vec{u}_{1,\Sigma_2}$ be the tuple of all Σ_2-variables, and $\vec{u}_1$ be the tuple of all shared variables.[6] Obviously, φ_1 is equivalent to the sentence

$$\exists \vec{u}_1 \left(\exists \vec{u}_{1,\Sigma_1} \ \gamma_{1,\Sigma_1} \wedge \exists \vec{u}_{1,\Sigma_2} \ \gamma_{1,\Sigma_2} \right).$$

The next two steps of the algorithm are nondeterministic, i.e., a given sentence is transformed into finitely many new sentences. Here the idea is that the original sentence is valid iff at least one of the new sentences is valid.

Step 3: Variable identification.

Consider all possible partitions of the set of all shared variables. Each of these partitions yields one of the new sentences as follows. The variables in each class of the partition are "identified" with each other by choosing an element of the class as representative, and replacing in the sentence all occurrences of variables of the class by this representative. Quantifiers for replaced variables are removed.

Let $\exists \vec{u}_2 \left(\exists \vec{u}_{1,\Sigma_1} \ \gamma_{2,\Sigma_1} \wedge \exists \vec{u}_{1,\Sigma_2} \ \gamma_{2,\Sigma_2} \right)$ denote one of the sentences obtained by Step 3, where $\vec{u}_2$ denotes the sequence of all representatives of shared variables.

Step 4: Choose signature labels and ordering.

We choose a label Σ_1 or Σ_2 for every (shared) variable in $\vec{u}_2$, and a linear ordering $<$ on these variables.

For each of the choices made in Step 3 and 4, the algorithm yields a pair (α, β) of sentences as output.

Step 5: Generate output sentences.

The sentence $\exists \vec{u}_2 (\exists \vec{u}_{1,\Sigma_1} \ \gamma_{2,\Sigma_1} \wedge \exists \vec{u}_{1,\Sigma_2} \ \gamma_{2,\Sigma_2})$ is split into two sentences

$$\alpha = \forall \vec{v}_1 \exists \vec{w}_1 \ldots \forall \vec{v}_k \exists \vec{w}_k \exists \vec{u}_{1,\Sigma_1} \ \gamma_{2,\Sigma_1}$$

and

$$\beta = \exists \vec{v}_1 \forall \vec{w}_1 \ldots \exists \vec{v}_k \forall \vec{w}_k \exists \vec{u}_{1,\Sigma_2} \ \gamma_{2,\Sigma_2}.$$

Here $\vec{v}_1 \vec{w}_1 \ldots \vec{v}_k \vec{w}_k$ is the unique re-ordering of $\vec{u}_2$ along $<$. The variables $\vec{v}_i$ ($\vec{w}_i$) are the variables with label Σ_2 (label Σ_1).

Thus, the overall output of the algorithm is a finite set of pairs of sentences. Note that the sentences α and β are positive formulae, but they need no longer be existential positive formulae.

[6] The order in these tuples can be chosen arbitrarily.

Correctness of the decomposition algorithm

If one of the output pairs is valid, then the original sentence is valid. But also, if the input sentence is valid, then there exists a valid output pair.

Proposition 3.4.3 *$\mathfrak{C}_1^{\Sigma} \models \varphi_0$ if and only if $\mathfrak{A}_1^{\Sigma_1} \models \alpha$ and $\mathfrak{A}_2^{\Sigma_2} \models \beta$ for some output pair (α, β).*

Obviously, Theorem 3.4.2 follows immediately.

3.4.2 The General Positive Case

The goal of this subsection is to show that the decomposition method introduced above can be extended such that it becomes possible to decide validity of *general* positive sentences in the free amalgamated product $\mathfrak{C}_1^{\Sigma} = \mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$. The main idea is to transform positive sentences (with arbitrary quantifier prefix) into existential positive sentences by Skolemising the universally quantified variables.[7] In principle, the decomposition algorithm for positive sentences is now applied twice to decompose the input sentence into three positive sentences α, β, ρ, whose validity must respectively be decided in $\mathfrak{A}_1^{\Sigma_1}$, $\mathfrak{A}_2^{\Sigma_2}$, and the absolutely free term algebra over the Skolem functions.

The extended decomposition algorithm

The input is a positive sentence φ_1 in the mixed signature $\Sigma_1 \cup \Sigma_2$. We assume that φ_1 is in prenex normalform, and that the matrix of φ_1 is in disjunctive normalform. The algorithm proceeds in two phases.

Phase 1: Via Skolemisation of universally quantified variables, φ_1 is transformed into an existential sentence φ_1' over the signature $\Sigma_1 \cup \Sigma_2 \cup \Gamma_1$. Here Γ_1 is the signature consisting of all the new Skolem function symbols that have been introduced.

Suppose that φ_1' is of the form $\exists \vec{u}_1 (\bigvee \gamma_{1,i})$, where the $\gamma_{1,i}$ are conjunctions of atomic formulae. Obviously, φ_1' is equivalent to $\bigvee (\exists \vec{u}_1\ \gamma_{1,i})$, and thus it is sufficient to decide validity of the sentences $\exists \vec{u}_1\ \gamma_{1,i}$. Each of these sentences is used as input for the decomposition algorithm.

The atomic formulae in $\gamma_{1,i}$ may contain symbols from the two (disjoint) signatures Σ_1 and $\Sigma_2 \cup \Gamma_1$. In Phase 1 we treat the sentences $\exists \vec{u}_1 \gamma_{1,i}$ by means of Steps 1–5 of the decomposition algorithm, finally splitting them into positive Σ_1-sentences α and positive $(\Sigma_2 \cup \Gamma_1)$-sentences φ_2. Thus, the output of Phase 1 is a finite set of pairs (α, φ_2).

[7] We are Skolemising *universally* quantified variables since we are interested in validity of the sentence and not in satisfiability.

Phase 2: In the second phase, φ_2 is treated exactly as φ_1 was treated before, applying Skolemisation to universally quantified variables and Steps 1–5 of the decomposition algorithm a second time. Now we consider the two (disjoint) signatures Σ_2 and $\Gamma = \Gamma_1 \cup \Gamma_2$, where Γ_2 contains the Skolem functions that are introduced by the Skolemisation step of Phase 2. We obtain output pairs of the form (β, ρ), where β is a positive sentence over the signature Σ_2 and ρ is a positive sentence over the signature Γ. Together with the corresponding sentence α (over the signature Σ_1) we thus obtain triples (α, β, ρ) as output.

For each of these triples, the sentence α is now tested for validity in $\mathfrak{A}_1^{\Sigma_1}$, β is tested for validity in $\mathfrak{A}_2^{\Sigma_2}$, and ρ is tested for validity in the absolutely free term algebra $\mathcal{T}(\Gamma, X)$ with countably many generators X, i.e., the free algebra over X for the class of all Γ-algebras.[8] We have seen that this structure is a quasi-free structure with atom set X (Examples 3.2.17 (3)).

Correctness of the extended decomposition algorithm

We must show that the original sentence φ_1 is valid iff for one of the output triples, all three components are valid in the respective structures. The – here not presented – proof depends on the following lemma, which exhibits an interesting connection between Skolemisation and free amalgamation with an absolutely free algebra.

Lemma 3.4.4 *Let $\mathfrak{A}_1^{\Sigma}$ be a quasi-free structure with atom set X, and let γ be a positive Σ-sentence. Suppose that the existential positive sentence γ' is obtained from γ via Skolemisation of the universally quantified variables in γ, introducing the set of Skolem function symbols Γ. Let $\mathfrak{A}_2^{\Gamma} := \mathcal{T}(\Gamma, X)$ be the absolutely free term algebra over Γ with generators X, and let $\mathfrak{C}_1^{\Sigma\cup\Gamma}$ be the free amalgamated product of $\mathfrak{A}_1^{\Sigma}$ and $\mathfrak{A}_2^{\Gamma}$. Then $\mathfrak{A}_1^{\Sigma} \models \gamma$ if, and only if, $\mathfrak{C}_1^{\Sigma\cup\Gamma} \models \gamma'$.*

Correctness of the extended decomposition algorithm is an easy consequence of this lemma.

Proposition 3.4.5 *$\mathfrak{C}_1^{\Sigma_1\cup\Sigma_2} \models \varphi_1$ if, and only if, there exists an output triple (α, β, ρ) such that $\mathfrak{A}_1^{\Sigma_1} \models \alpha$, $\mathfrak{A}_2^{\Sigma_2} \models \beta$, and $\mathcal{T}(\Gamma, X) \models \rho$, where Γ consists of the Skolem functions introduced in Phases 1 and 2 of the algorithm.*

The proposition shows that decidability of the positive theory of the free amalgamated product $\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ can be reduced to decidability of the positive theories of $\mathfrak{A}_1^{\Sigma_1}$, $\mathfrak{A}_2^{\Sigma_2}$, and of an absolutely free term algebra $\mathcal{T}(\Gamma, X)$. It is well-known that the whole first-order theory of absolutely free term algebras is decidable [33, 70, 72]. Thus, Theorem 3.4.1 follows immediately. In connection with the Theorems 3.3.12 and 3.3.14, the following generalisation is obtained.

[8] Note that Γ contains no predicate symbols.

Theorem 3.4.6 *If* $(\mathfrak{A}_1^{\Sigma_1}, X), \ldots, (\mathfrak{A}_n^{\Sigma_n}, X)$ *are quasi-free structures over disjoint signatures, then the full positive theory of the free simultaneous amalgamated product* $\mathfrak{A}_1^{\Sigma_1} \otimes \cdots \otimes \mathfrak{A}_n^{\Sigma_n}$ *is decidable, provided that the positive theories of all structures* $\mathfrak{A}_i^{\Sigma_i}$ *are decidable* $(1 \leq i \leq n)$*.*

3.5 Conclusion

This chapter's purpose was to introduce fundamental concepts of combining constraint systems. We presented the notion of a quasi-free structure exploring its algebraic and logical properties. Quasi-free structures comprise many important non-numerical infinite solution domains for constraint solving such as quotient term algebras, rational tree algebras, vector space, sets, multisets and lists and certain feature structures. We discussed the properties a suitable combination of structures should have, namely sharing relevant structural properties with the components and being rather general. We introduced the free amalgamated product of two structures which is characterised by being the most general combination of two quasi-free structures and gave an explicit construction how to obtain the free amalgamated product for arbitrary quasi-free structures. Finally we drew our attention to the combination of constraint solvers presenting a non-deterministic algorithms to reduce the solving of mixed constraints over the joint signatures to solving of pure constraints in the components. By means of correctness of this algorithm we showed that the positive theory of the free amalgamated product is decidable, provided the positive theories of the component quasi-free structures are decidable.

On this given base, it is three different aspects of combining constraint systems that we would like to investigate in the next chapters. Firstly, the decomposition algorithm introduced is meant to be clear and simple. In its current form, it is well suited for explaining the method and proving its correctness. But it is highly non-deterministic, and hence totally unsuitable for implementation. We will present the search space spanned by the non-deterministic steps of the algorithm underpinning thereby the imminent need for optimisations and explore systematic, generally applicable optimisation methods. We will see that there are two principled ways to reduce the non-determinism which together can shrink the search space for certain input problems by several orders of magnitude.

Another aspect faces the amalgamation construction. The free amalgamated product has the nice characterising property of being the most general combination of quasi-free structures. But is it the only general combination construction? We will see that there is another combination, namely rational amalgamation, which constitutes a very general combined solution domain for a large class of quasi-free structures. In opposite to free amalgamation, rational amalgamation allows an infinite number of switchings from one component to the other in the elements of the combined solution domain, and therefor permits the solution of mixed cyclic equations or constraints.

Finally, we investigate to what extend negation can be handled in combination.

The constraints in the last chapter will be literals, atoms or negated atoms, while they are just (positive) atoms in the other chapters. We will find the free amalgamated product to be a combined solution domain suitable for combining mixed positive and negative constraints. Indeed, mixed constraint over the joint signatures of the components can be solved in the free amalgamated product, if the pure constraints together with some technical requirements can be solved in the quasi-free component structures. We will also take a look at the independence property of negative constraints. For a given structure, the independence property states that a conjunction of negative constraints is solvable, if each negative constraint is solvable in isolation, ignoring the others. This is quite a useful property in actual constraint solving. We explore general properties that a quasi-free structure must have in order to own the independence property. And we also derive a modularity result stating under which conditions the independence property of two component quasi-free structures is inherited by their free amalgamated product.

Chapter 4

Optimisation Techniques

4.1 Introduction

The previous chapter described a fairly general combination algorithm for constraint solvers for quasi-free structures over disjoint signatures. For reasons of expository clarity, and also to simplify correctness proofs, no effort was made in trying to lay out the algorithm in an *efficient* way. The part of the algorithm that introduces the complexity are the three non-deterministic steps of variable identification, labelling and ordering. The search space spanned by the combination of all different choices that can be made in these three steps is huge; it is indeed that huge that a naive implementation of the algorithm is in praxi deemed to non-termination even for small input problems. In a subsequent subsection on the complexity of the algorithm, we will show this in detail.

The consequence of this simple observation is obviously, that any implementation of the algorithm must employ optimisation techniques. In principle, there are two different ways to handle the problem. For the task of combining two particular given constraint systems, one could start by defining a special combined solution domain and then develop a very specific decomposition algorithm for combining the two constraint solvers at hand. This may be the solution of choice, if for a concrete implementation speed is more important than anything else.

The line we would like to follow here is a one that is interested in general optimisation techniques applicable to a large number of constraint systems. Especially, we want to profit from general results presented in the previous chapter. Therefore we choose to keep the free amalgamated product as combined solution domain. And we take its decomposition algorithm as a starting point for our optimisation rather then developing a new one from scratch. In this way, we may not be able to provide the most efficient combination algorithm for two specific constraint solvers – and it is important to see that we do not claim this and hence do not compete with specific combination algorithms – but provide strategies that are usable in many circumstances and allow the integration of several diverse constraint solvers in a suitable amount of time without being forced to redesign everything.

In principle, there are two different general methods of optimisation we will present. The first one is called the *iterative* method. It is based on the insight that in a combination of many constraint solvers it is not wise to make *all* non-deterministic choices for all components first and only then simultaneously test solvability in all components. It turns out to be more sensible to restrict attention to one component at a time and find a set of non-deterministic choices for which the component solver can solve its subproblem. Though this seems clear, the difficulty in this method lies in showing that neither correctness nor completeness of the combination algorithm is lost, something that is far from obvious. This method's use is meaningful only, when more than two constraint solvers are combined, and with a growing number of components it shows its full strength.

The second optimisation method we will present is called the *deductive* method. The underlying observation here is that many choices need not be made *non*-deterministically. The input problem and the component constraint systems frequently enforce certain decisions to be made in a particular fashion, because otherwise the problem would be plain unsolvable. These choices can be made deterministically, and what is more, choices made in a particular fashion in line with demands of one component can trigger new deterministic decisions to keep subproblems of other components solvable. Hence one needs new component constraint solvers that are capable of deducing what decisions can be made deterministically on the base of their subproblem and the choices made so far. And one needs a new combination algorithm that consults the component solvers after a non-deterministic decision was made to find out which deterministic ones it involves and uses constraint propagation techniques to circulate decisions between the component solvers. The impact of this method is enormous. It turns out that in many cases there is enough information available in the constraint systems to shrink the non-deterministic search space by orders of magnitude.

This chapter heavily relies on [61]. The optimisation methods described above were originally designed for the combination of equational unification algorithm; and it is that, what the technical report describes. But these methods can be applied to the more general case of combining constraint solvers straight forwardly. The original combination algorithm for combining constraint solvers, as described in the previous chapter, differs from the one for combining unification algorithms (as presented in [14]) only in the obvious way. Every predicate different from equality must be purified (i.e., alien subterms must be abstracted away) and assigned to the component the signature of which it belongs to. Thus only a small extension in the purification step is needed, the three main steps, the non-deterministic guessing of the variable identification, labelling and ordering remain the same.

On the terminological side, we replaced the notion *constraint* as used in [61] by the notion *decision*. Since we are describing the more general case of combining constraint solvers and not just equational unification algorithms, we would end up using the word "constraint" for both the input problem and the individual non-deterministic choices, and that would only cause a lot of confusion.

The optimisation methods and the above cited report are a co-production with J. Richts. He developed and implemented the deductive method and the deductive component algorithms for equational unification. All these are described in detail in his forthcoming doctoral dissertation. The author's contribution is the development of the iterative method and the integration of the two methods. He also implemented the deductive component algorithm for feature structures.

The first section of this chapter shortly reviews the original combination algorithm and explains its complexity. It also presents some basic optimisations that are obvious or long known and should be taken into account by any implementation. The second section introduces the concept of a *decision* as a technical term thereby laying out a common framework for describing both methods. The following two sections are devoted to introducing and describing our two optimisation methods. We will thereafter show that it is easy to integrate them into a common system and present some run time results to empirically support our theoretical claims. Finally, since Boudet [20, 21] developed optimisation techniques for combining unification algorithms that show certain similarities to our work, we spend some time to detail the differences and similarities of the two approaches.

In this chapter, a constraint problem is a conjunction of atomic formulae. Variables occurring therein are implicitly existentially quantified.

4.2 The Base for Optimisation

The Original Combination Algorithm

In this section, we briefly recapitulate the original combination algorithm as described in the previous chapter while at the same time extending it from 2 to n constraint solvers that are combined simultaneously. For $i = 1, \dots, n$ ($n \geq 2$), let Σ_i be pairwise disjoint signatures and $\Sigma := \bigcup_{i=1}^{n} \Sigma_i$. The input problem Γ is a conjunction of atomic constraints over Σ. We say that Γ is in *decomposed form*, if Γ has the form $\bigcup_{i=1}^{n} \Gamma_i$ where each Γ_i is a pure constraint problem of component i over the signature Σ_i. Any constraint problem Γ can be transformed into a constraint problem in decomposed form that is solvable, iff the original problem is solvable, by a simple deterministic preprocessing step, namely variable abstraction. In the following, we will therefore always assume that a constraint problem is in decomposed form $\bigcup_{i=1}^{n} \Gamma_i$.

The combination algorithm consists of three non-deterministic steps which result in a linear constant restriction for the constraint problem. The notion of a linear constant restriction is introduced for unification problems at the end of Section 2.2. Let Γ be a constraint problem in decomposed form.

Step 1: Variable identification Non-deterministically choose a partitioning Π of $Var(\Gamma)$ and a representative for each class. In all constraints, replace each variable by its representative. We obtain a new formula $\Gamma' := \bigwedge \Gamma'_i$. Let Y be the set of representatives.

Step 2: Labelling Non-deterministically choose a labelling function $Lab : Y \rightarrow \{\Sigma_1, \ldots, \Sigma_n\}$.

Step 3: Ordering Non-deterministically choose a strict linear order $<_L$ on the variables Y.

Step 4: Component solvers For $i = 1, \ldots, n$, form constraint problems with constant restrictions: in Γ_i treat each $x \in Y$ with $Lab(x) \neq \Sigma_i$ as a free constant and use the linear constant restrictions induced by $<_L$.

Theorem 4.2.1 *The input problem Γ has a solution in the free amalgamated product, if and only if there exists an output tuple $(\Gamma'_1, \ldots, \Gamma'_n)$ with Lab and $<_L$ in Step 4 such that for $i = 1, \ldots, n$ the constraint problem with linear constant restriction $(\Gamma'_i, (Lab, <_L))$ has a solution.*

A proof of this theorem can be found in [15]. For clarity, Let us define the notion of a solution for a constraint problem with linear constant restriction in one of the components.

Definition 4.2.2 Let $(\mathfrak{A}_i^{\Sigma_i}, X)$ be a quasi-free structure. Let $L = (Lab, <_L)$ be a linear constant restriction, and Γ_i a constraint problem over signature Σ_i. A substitution σ is a called a *solution* of the constraint problem with linear constant restriction (Γ_i, L), iff it is a solution of Γ_i and for every variable $x \in \mathrm{dom}(Lab)$ with $Lab(x) \neq \Sigma_i$ holds $\sigma(x) \in X$, and for all variables $x, y \in \mathrm{dom}(Lab)$ with $Lab(x) = \Sigma_i, Lab(y) \neq \Sigma_i, x <_L y$ holds $\sigma(y) \notin \mathrm{Stab}^{\mathfrak{A}_i}(\sigma(x))$.

If the constraint problem is positive, i.e., contains no negation, as we assume in this chapter, then constraint problems with linear constant restrictions can be translated into purely logic problems, as is shown in Lemma 3.2.29.

Complexity of the Original Combination Algorithm

The above algorithm contains three non-deterministic steps. By guessing the right variable identification, labelling and ordering, one can find a solution in polynomial time. Hence the algorithm belongs to the complexity class *nondeterministic-polynomial time (NP)*. But any implementation has to be deterministic, thus the above classification is not really satisfactory. We will now give a bound for the search tree spanned by the combinations of different choices that can be made. Suppose that there are n component systems and k variables.[1]

[1] The following basic combinatorial notions and formulae can be found in any book on this subject. A particular constructive approach that we consulted is [105].

In Step 1, we have to calculate an upper bound for all partitions of k Variables. The number of partitions of a k element set is known as the k-th Bell number B_k. It can be recursively calculated by the formula

$$B_0 = 1, \qquad B_{k+1} = \sum_{r=0}^{k} \binom{k}{r} B_r \; .$$

But there is a more appropriate way for our purposes. The choices in Steps 2 and 3 are not independent of the choices in the first step, they depend on the number of remaining representatives. The number of ways to partition a k element set into r partitions is called the Sterling number of the second kind. It is defined by

$$s_{k,r} := \frac{1}{r!} \sum_{j=0}^{r} (-1)^{r-j} \binom{r}{j} j^k$$

and can be recursively calculated by

$$s_{k,r} = s_{k-1,r-1} + r \cdot s_{k-1,r} \; .$$

Now, the k-th Bell number

$$B_k = \sum_{r=1}^{k} s_{k,r}$$

by definition of Bell numbers and Sterling numbers of the second kind.

Let r be the number of representatives remaining after variable identification in the first step. In Step 2, the labels for the variables (after identification) are chosen independent of each other. Thus there are

$$n^r$$

different variable labellings.

In Step 3, we are looking for all linear orders of the representatives. Thus we have to consider all permutations of a r element sequence, and it is known that there are

$$r!$$

many.

Hence, the size of the search space is given by

$$\sum_{r=1}^{k} s_{k,r} \cdot n^r \cdot r! \; .$$

A deterministic implementation of the algorithm is in the complexity class *deterministic (singly-) exponential time*. To gain an intuition on the size of the search space, consider a small input problem with 3 component systems and 5 variables. Then there are exactly 52923 different leaf nodes in the search tree. So, even small problems are practically intractable. This obviates the need for optimisation techniques.

Basic Optimisations

The following optimisations are very straightforward and should be taken into account in any implementation. They have already been discussed by other authors or are obvious. We mention them here for the sake of completeness.

Only variables occurring in more than one component constraint Γ_i have to be considered by the combination algorithm. Hence we define the set of *combination variables* or *shared variables* $\mathcal{U} := \{x \mid \exists i, j : i \neq j, x \in Var(\Gamma_i) \cap Var(\Gamma_j)\}$ and use $\mathcal{U}_i := \mathcal{U} \cap Var(\Gamma_i)$ to denote the set of combination variables of constraint Γ_i. Only combination variables need to be considered in the non-deterministic steps. Because if L is a linear constant restriction containing only combination variables such that (Γ, L) is solvable, then L can be extended to a linear constant restriction L' with all variables that has an identical set of solutions. This fact was discovered independently by Boudet [20, 21] and Baader & Schulz [5].

Secondly, two different linear orderings $<_{L_1}$ and $<_{L_2}$ may lead to the same constraint problems with linear constant restrictions. Suppose that the orderings $<_{L_1}$ and $<_{L_2}$ differ only in the order of two variables with identical label which are adjacent w.r.t. $<_{L_1}$ and $<_{L_2}$. Then the restrictions on atoms that can occur in the stabilisers of other elements induced by these orderings are identical and the constraint problem with linear constant restriction obtained from $<_{L_1}$ is solvable iff the problem obtained from $<_{L_2}$ is solvable. Thus the algorithm does not need to consider the ordering of variables with identical label if no variable with different label lies between them in the chosen ordering. The orderings we will use in the following are therefore only *quasi-linear*: each two variables which have different labels must be ordered.

Thirdly, the set of pure constraint problems can be partitioned in such a way that each class of constraint problems can be solved independently of the others by the following. We call two pure constraint problems interrelated iff they share a variable. A class of constraint problems is a connected component with respect to this "interrelated"-relation. If two constraint problems belong to different classes, they do not even indirectly share variables. Thus they are totally independent of each other and can therefore be solved independently. Hence, we will assume that the constraint problem we must solve consists of just one class of interrelated constraint problems.

We generalise the notion of a linear constant restriction to respect the first two optimisations mentioned above and to include the variable identification given in the first step of the combination algorithm. A *generalised linear constant restriction* over a set of variables $\mathcal{U}$ is a triple $L = (\Pi, Lab, <_L)$ where Π is a partition of $\mathcal{U}$, $Lab : \mathcal{U} \to \{\Sigma_1, \ldots, \Sigma_n\}$ is a labelling function, and $<_L$ is a partial ordering of $\mathcal{U}$ with the following properties (we use $\equiv_\Pi$ to denote the equivalence relation induced by Π):

- Lab obeys the identification induced by Π ($Lab(x) = Lab(y)$ if $x \equiv_\Pi y$),
- $<_L$ obeys Π ($x' <_L y'$ if $x' \equiv_\Pi x$, $y' \equiv_\Pi y$, and $x <_L y$), and

- each two variables with different labels are ordered in $<_L$ ($x <_L y$ or $y <_L x$ if $Lab(x) \neq Lab(y)$).

A substitution σ solves (Γ_i, L) in the quasi-free structure $(\mathfrak{A}_i^{\Sigma_i}, X)$, iff σ solves Γ_i, and for all $x, y \in \mathcal{U}$

- $\sigma(x) = \sigma(y)$ if $x \equiv_\Pi y$,
- $\sigma(x) \in X$ whenever $Lab(x) \neq \Sigma_i$, and
- $\sigma(y) \notin \mathrm{Stab}^{\mathfrak{A}_i}(\sigma(x))$ whenever $Lab(y) \neq Lab(x) = \Sigma_i$ and $x <_L y$.

By item two, all variables that receive a label different from Σ_i are treated as constants by σ. By item three, the use of these constants in σ is further restricted. Two generalised linear constant restrictions L_1 and L_2 over $\mathcal{U}$ are called *equivalent*, if they have identical partitions and labelling functions and their orders differ at most in ordering variables of identical label. This definition induces an equivalence relation on all generalised linear constant restrictions for a given set of variables $\mathcal{U}$. If L_1 and L_2 are equivalent and a substitution σ solves (Γ, L_1), then σ also solves (Γ, L_2).

Proposition 4.2.3 *The input problem Γ has a solution in the free amalgamated product, if and only if there exists a generalised linear constant restriction $L = (\Pi, Lab, <_L)$ over $\mathcal{U}$, the combination variables of Γ, such that for the output tuple $((\Gamma_1, L), \ldots, (\Gamma_n, L))$ each constraint problem with generalised linear constant restriction (Γ_i, L) has a solution.*

It is clear that an optimised algorithm will compute just one generalised linear constant restriction for each equivalence class.

4.3 Decision Sets

The original algorithm makes all non-deterministic decisions first, and only thereafter it calls the component algorithms to determine whether the input problem with the thus chosen constant restriction is solvable. Our optimisations interleave these two parts. Hence we have to deal with linear constant restrictions which are only partially specified, i.e., restrictions representing the choices already made but making no statements about the decisions still open. In order to describe these partial constant restrictions and to have a common framework for describing our optimisations on a formal level, we introduce the notion of decision sets. Each element in such a set describes a single non-deterministic decision. There exist five different types of decisions.

Definition 4.3.1 Let $\mathcal{U}$ be the set of variables. A *decision* is an expression of the form $x \doteq y$, $x \mathrel{\dot{\neq}} y$, $x \mathrel{\dot{\leq}} y$, $x \mathrel{\dot{\mapsto}} \Sigma_i$, or $x \mathrel{\dot{\not\mapsto}} \Sigma_i$, where $x, y \in \mathcal{U}$ and $1 \leq i \leq n$. The decision $x \mathrel{\dot{<}} y$ is used as an abbreviation for $x \mathrel{\dot{\leq}} y, x \mathrel{\dot{\neq}} y$. With $\mathcal{L}(\mathcal{U})$ we denote the decision language.

Sets of decisions (for a set of variables $\mathcal{U}$) are – as usual – read conjunctively. In order to represent the two options when making a decision, we define the negation of a decision.

Definition 4.3.2 Let d be a decision. Its *negation* $\neg d$ is defined as follows:

$$\begin{array}{llll} \neg x \doteq y & := x \dot{\neq} y, & \neg x \dot{\neq} y & := x \doteq y, \\ \neg x \dot{\mapsto} \Sigma_j & := x \dot{\not\mapsto} \Sigma_j, & \neg x \dot{\not\mapsto} \Sigma_j & := x \dot{\mapsto} \Sigma_j, \\ \neg x \dot{\leq} y & := y \dot{<} x. & & \end{array}$$

These rules of negation reflect the three non-deterministic steps of the algorithm: Two variables have to be identified or treated as different variables; each variable has to be treated as a variable or like a constant in a particular component constraint problem; and two variables with distinct labels have to be ordered in one way or the other. In the following we formally define this correspondence between sets of decisions and linear constant restrictions.

Definition 4.3.3 Let $\mathcal{U}$ be a set of variables. A generalised linear constant restriction $L = (\Pi, Lab, <_L)$ over $\mathcal{U}$ *satisfies* a decision set D, if the following holds:

$$\begin{array}{rlrl} x \equiv_\Pi y & \text{if} \quad x \doteq y \in D, & x \not\equiv_\Pi y & \text{if} \quad x \dot{\neq} y \in D, \\ Lab(x) = \Sigma_i & \text{if } x \dot{\mapsto} \Sigma_i \in D, & Lab(x) \neq \Sigma_i & \text{if } x \dot{\not\mapsto} \Sigma_i \in D, \\ x <_L y \text{ or } x \equiv_\Pi y & \text{if} \quad x \dot{\leq} y \in D. & & \end{array}$$

The set of linear constant restrictions satisfying D is denoted by $\mathcal{L}(D)$. Two sets of decisions D_1 and D_2 are *equivalent* if $\mathcal{L}(D_1) = \mathcal{L}(D_2)$. A set D is called *inconsistent* if $\mathcal{L}(D) = \emptyset$.

So, the decisions are interpreted by a generalised linear constant restriction in a straightforward way.

Definition 4.3.4 A decision set D is called *closed* if

$$D = \{d \mid \text{ every } L \in \mathcal{L}(D) \text{ satisfies } \{d\}\}.$$

This definition implies that for each decision set D there is exactly one closed set which is equivalent to D; this set is called the *closure* of D. This closure can be computed efficiently; one has to consider that $\doteq$ denotes a congruence, $\dot{<}$ stands for an ordering, and $x \dot{\mapsto} \Sigma_i$ represents a functional relation. For example, a closure always contains $x \doteq x$ for all variables $x \in \mathcal{U}$, the two decisions $x \doteq y \in D$ and $y \dot{<} z \in D$ imply that $x \dot{<} z$ is in the closure of D, and the closure of $\{x \dot{\mapsto} \Sigma_i\}$ contains $x \dot{\not\mapsto} \Sigma_j$ for all $i \neq j$. In the following we will always assume that sets of decisions are closed, i.e., when adding decisions to a set we assume that the closure is formed immediately. With $\mathcal{C}_{\mathcal{L}(\mathcal{U})}$ we denote the set of all closed decision sets. There are two special closed decision sets. The first one is the closure of the empty decision set, denoted $C_\bot$. The second one is the closure of any inconsistent set, namely $C_\top = \mathcal{L}(\mathcal{U})$.

We have the following syntactic characterisation of closure.

Lemma 4.3.5 *Let D be a set of decisions over variables $\mathcal{U}$. D is closed, iff for all $x, y, z \in \mathcal{U}$*

$$
\begin{array}{l}
x \doteq x \in D, \\
x \doteq y \in D \Longrightarrow y \doteq x \in D, \\
x \doteq y, y \doteq z \in D \Longrightarrow x \doteq z \in D, \\
x \dot{\neq} y \in D \Longrightarrow y \dot{\neq} x \in D, \\
x \doteq y, y \dot{\neq} z \in D \Longrightarrow x \dot{\neq} z \in D, \\
x \dot{\mapsto} \Sigma_i \in D \Longrightarrow x \dot{\not\mapsto} \Sigma_j \in D \textit{ for all } j \neq i, \\
x \doteq y, x \dot{\mapsto} \Sigma_i \in D \Longrightarrow y \dot{\mapsto} \Sigma_i \in D, \\
x \dot{\mapsto} \Sigma_i, y \dot{\mapsto} \Sigma_j \in D, i \neq j \Longrightarrow x \dot{\neq} y \in D, \\
x \dot{<} y, y \dot{<} z \in D \Longrightarrow x \dot{<} z \in D, \\
x \doteq y, y \dot{<} z \in D \Longrightarrow x \dot{<} z \in D, \\
x \dot{<} y, y \doteq z \in D \Longrightarrow x \dot{<} z \in D, \\
x \dot{<} y \in D \Longrightarrow x \dot{\neq} y \in D.
\end{array}
$$

Proof. It is straight forward to check that if D is closed all of the above conditions must hold. Thus the only interesting part is to see that the conditions characterise closure. So let D fulfil these conditions. We have to show that every decision that is satisfied by all linear constant restrictions $L \in \mathcal{L}(D)$ is already contained in D. Decisions satisfied by all $L \in \mathcal{L}(D)$ are those ones that follow from the definition of a generalised linear constant restriction.

Let $L = (\Pi, Lab, <_L)$ be a generalised linear constant restriction. The first part, Π, is a partition of $\mathcal{U}$. It interprets $\doteq$ and $\dot{\neq}$. Thus $\doteq$ is always reflexive, symmetric and transitive in the closure of D, $\dot{\neq}$ is symmetric and $\doteq$-closed. Hence the first five conditions.
The second component, Lab, is a labelling function that obeys Π. Condition six corresponds to function-hood and conditions seven and eight reflect that Lab respects Π.
The third component, $<_L$, is a partial order that obeys Π and orders each two variables with different labels. Condition nine give the transitivity of the partial order. Conditions ten to twelve reflect that $<_L$ obeys Π. Since the definition of $<_L$ only demands that two variables of different labels be ordered, but not the way they are, different linear constant restrictions may order them differently. Hence there can't be any decision satisfied by all linear constant restrictions in $\mathcal{L}(D)$ resulting from that part of the definition of $<_L$. Since this is all the definition demands there are no more conditions required. ∎

We need a criterion when a set of decisions already represents one linear constant restriction constructed by the algorithm, i.e., when no more non-deterministic decisions have to be made.

Definition 4.3.6 A set of decisions D is *complete*, if all generalised linear constant restriction in $\mathcal{L}(D)$ are equivalent.

From this definition and the one above it follows that there is a one-to-one correspondence between the equivalence classes of generalised linear constant restrictions over $\mathcal{U}$ and closed and complete sets of decisions for $\mathcal{U}$.

In order to test inconsistency and completeness in the algorithm, we need a syntactic formulation of these properties. This is provided by the following lemma.

Lemma 4.3.7

1. *A closed set of decisions D is inconsistent iff $d \in D$ and $\neg d \in D$ for some decision d.*
2. *A closed and consistent set of decisions D (for variables $\mathcal{U}$) is complete iff for all $x, y \in \mathcal{U}$*

$$\begin{array}{l} \textit{either}\ \ x \doteq y \in D \ \ \textit{or}\ \ x \not\doteq y \in D, \ \ \textit{and} \\ \textit{either}\ \ x \dot{<} y \in D \ \ \textit{or}\ \ y \dot{<} x \in D \quad \textit{if } x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i \in D, \ \textit{and} \\ x \dot{\mapsto} \Sigma_i \in D \ \textit{for one}\ \Sigma_i. \end{array}$$

Proof. The characterisation of consistency is obviously correct.

Let D be a consistent closed and complete set. Since all generalised linear constant restrictions in $\mathcal{L}(D)$ are equivalent, all their partitions and labelling functions are identical. As the partition and labelling function are total on $\mathcal{U}$, clearly for all $x, y \in \mathcal{U}$: either $x \doteq y \in D$ or $x \not\doteq y \in D$ and there is an $i \leq n$ with $x \dot{\mapsto} \Sigma_i \in D$ by D being closed.
All generalised linear constant restrictions in $\mathcal{L}(D)$ order all pairs of variables with different label, and since they are equivalent, their ordering information on those variables that differ in label are identical. Thus for all $x, y \in \mathcal{U}$ with $x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i$ either $x \dot{<} y \in D$ or $y \dot{<} x \in D$, because D is closed.

Suppose D is consistent and closed and fulfils the three conditions. Let $L_1 = (\Pi_1, Lab_1, <_{L_1})$ and $L_1 = (\Pi_2, Lab_2, <_{L_2})$ be two generalised linear constant restrictions that satisfy D. For all $x, y \in \mathcal{U}$ either $x \doteq y \in D$ or $x \not\doteq y \in D$. Therefore for all $x, y \in \mathcal{U}$ either $x \equiv_{\Pi_1} y$ and $x \equiv_{\Pi_2} y$ or $x \not\equiv_{\Pi_1} y$ and $x \not\equiv_{\Pi_2} y$. Thus $\Pi_1 = \Pi_2$.
For all $x \in \mathcal{U}$ there is an i such that $x \dot{\mapsto} \Sigma_i \in D$. Therefore for all $x \in \mathcal{U}$: $Lab_1(x) = Lab_2(x) = \Sigma_i$. And thus $Lab_1 = Lab_2$.
For all $x, y \in \mathcal{U}$ such that there is an i with $x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i \in D$ either $x \dot{<} y \in D$ or $y \dot{<} x \in D$. Thus either $x <_{L_1} y$ and $x <_{L_2} y$ or $y <_{L_1} x$ and $y <_{L_2} x$. Therefore $<_{L_1}$ and $<_{L_2}$ order all pairs of variables that differ in label identical. Therefore, L_1 and L_2 are equivalent and D is complete. ∎

Definition 4.3.8 Let $(\mathfrak{A}_i^{\Sigma_i}, X)$ be a quasi-free structure over signature Σ_i. A *constraint problem with decision set* is a finite set of atomic Σ_i-constraints Γ (read conjunctively) together with a set of decisions D. A substitution σ is a *solution* of (Γ, D), if σ is a solution of Γ and for all variables $x, y \in \mathcal{U}$

- if $x \doteq y \in D$ then $\sigma(x) = \sigma(y)$,
- if $x \not\doteq y \in D$ then $\sigma(x) \neq \sigma(y)$,
- if $x \not\dot{\mapsto} \Sigma_i \in D$ then $\sigma(x) \in X$,
- if $x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i, x \dot{<} y \in D$ then $\sigma(y) \notin \mathrm{Stab}^{\mathfrak{A}_i}(\sigma(x))$.

4.4 Iterative Decomposition

The Principle

A major disadvantage of the original method is late detection of failure. Suppose the input problem consists of constraint problems of five different components and that the second sub constraint problem – and thus the whole problem – is unsolvable. The original method always makes all decisions for all constraint problems. In order to detect the insolvability of the second component, all decisions for all the following components must be considered as well before testing solvability. Thus the whole search tree of the remaining constraint problems must be considered before the algorithm establishes that at any leaf of this tree the second component is unsolvable, independently of the decisions made for later components.

Avoiding this problem is the main goal of the iterative decomposition method: components are solved iteratively, one component at a time. All decisions in the non-deterministic steps are made locally, for the current component only, and after that, this component is tested for solvability. So we start by non-deterministically choosing a variable identification, a labelling, and an ordering that solves the first component problem. And we proceed from one component constraint problem to another by making the choices necessary to solve the next component problem while respecting previously made choices. If it turns out that previously made choices make the current component problem unsolvable, we have to backtrack to the previous component problem and try another set of choices. Making choices locally just for one component problem means the following. We identify or discriminate variables of the current component problem, only. We label variables of the current component problem, and furthermore we only determine whether a variable receives the signature of the current component problem as label or whether it is treated as a constant in this component. And just the variables of the current component problem are ordered.

The advantages of the iterative decomposition are twofold. Firstly, iterative decomposition remedies the disadvantage of late detection of insolvability as described above. If a component problem is unsolvable, this is detected when trying to solve this component problem. Therefore no decisions about later component problems will be made.

Secondly, the search space is reduced as compared to the original algorithm by avoiding certain superfluous choices. Even under the assumption that all component problems of the input constraint problem are interrelated, there are variable identifications and orderings that are not needed. For example, if two variables do not occur commonly in one component problem after all identifications being made, then ordering them either way does not affect solvability. Since iterative decomposition can make decisions only on variables that occur together in at least one component problem, these superfluous choices will not be made.

The Algorithm

Before we present the algorithm, we have to define a condition when all choices for one component constraint problem have been made. Recall that $\mathcal{U}_i$ denotes the set of combination variables of problem Γ_i.

Definition 4.4.1 A decision set D is *complete for component* i, iff for all variables $x, y \in \mathcal{U}_i$

$$\begin{array}{lll} \text{either} \quad x \doteq y \in D \text{ or } \quad x \not\doteq y \in D, & \text{and} \\ \text{either } x \dot{\mapsto} \Sigma_i \in D \text{ or } x \not\dot{\mapsto} \Sigma_i \in D, & \text{and} \\ \text{either} \quad x \dot{<} y \in D \text{ or } \quad y \dot{<} x \in D & \text{if } x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i \in D. \end{array}$$

In the following description, we collect previously made decisions in the form of sets of decisions D_i. Each set D_i will be a consistent closed set of decisions collecting the choices we have made so far. Define $D_0 := \mathit{Clo}(\emptyset)$, i.e., the initial set of decisions is trivial.

For component problems $i := 1$ to n repeat the following steps

Step 1: Variable Identification

Choose a partition Π amongst the variables $\mathcal{U}_i$. Define $D'_{i,=} := \{x \doteq y \mid x \equiv_\Pi y\}$ and $D'_{i,\neq} := \{x \not\doteq y \mid x \not\equiv_\Pi y\}$. The partition Π must be chosen in such a way, that $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ is consistent. This means that previously made identifications and discriminations must be observed.

Step 2: Labelling

Choose some set $V \subseteq \mathcal{U}_i$ to form the labelling decision set $D'_{i,Lab} := \{x \dot{\mapsto} \Sigma_i \mid x \in V\} \cup \{x \not\dot{\mapsto} \Sigma_i \mid x \in \mathcal{U}_i \setminus V\}$ in such a way that $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab}$ is consistent. Therefore labels are assigned to whole classes of the partition Π, and a label can only be assigned to variables that have not yet received one.

Step 3: Ordering

Choose a set of ordering decisions $D'_{i,<} \subset \{x \dot{<} y, y \dot{<} x \mid x, y \in \mathcal{U}_i \text{ and } x \dot{\mapsto} \Sigma_i, y \not\dot{\mapsto} \Sigma_i \in D'_{i,Lab}\}$ such that each pair $x, y \in \mathcal{U}_i$ with distinct labels is ordered and $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab} \cup D'_{i,<}$ is consistent. This implies amongst other things that the order is non-cyclic and that previous ordering decisions are respected.

Define D_i as the closure of $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab} \cup D'_{i,<}$. Define $D_i|_{\mathcal{U}_i} \subseteq D_i$ as the subset of D_i that contains only decisions over the variable set $\mathcal{U}_i$.

Step 4: Testing the Component Problem Γ_i with Decision Set
If there is a Σ_i-substitution that solves $(\Gamma_i, D_i|_{\mathcal{U}_i})$, continue with the next component problem. Otherwise choose another set of decisions. If no other choice is left for the current component problem Γ_i, backtrack over components $i-1, \ldots, 1$, i.e., try another choice in the preceding components.

Proposition 4.4.2 *The input problem Γ is solvable, iff there is a set D_n such that for each $i = 1, \ldots, n$ the component problem with decision set $(\Gamma_i, D_i|_{\mathcal{U}_i})$ is solvable.*

Note that testing $(\Gamma_i, D_i|_{\mathcal{U}_i})$ for solvability can be performed by the same component algorithms as are used in the original algorithm.

It is a quite subtle task to enumerate all the possible consistent extensions of a given decision set without relying on an inefficient generate and test method. For the first two steps, this task is not too difficult. For the third step, the description of such an enumeration algorithm is rather involved. It follows in Section 4.5. We will now give syntactic criteria for when an extension is consistent.

Lemma 4.4.3 *In Step 1, $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ is consistent, iff the following two conditions are true: both $x \equiv_\Pi y$ if $x \doteq y \in D_{i-1}$, and $x \not\equiv_\Pi y$ if $x \not\doteq y \in D_{i-1}$ for all $x, y \in \mathcal{U}_i$.*

Proof. If $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ is consistent, then clearly the two conditions hold. For the inverse direction, D_{i-1} and $D'_{i,=} \cup D'_{i,\neq}$ are consistent. So the only way inconsistencies can arise by $d \in D_{i-1}$ and $\neg d \in D'_{i,=} \cup D'_{i,\neq}$ for some decision d. This can only happen by either $x \doteq y \in D_{i-1}$ and $x \not\equiv_\Pi y$ or $x \not\doteq y \in D_{i-1}$ and $x \equiv_\Pi y$ for some $x, y \in \mathcal{U}_i$. ■

Lemma 4.4.4 *In Step 2, $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab}$ is consistent, iff the following two conditions are true: both $[x]_\Pi \subseteq V$ for all $x \in V$, and $V \cap \{x \mid \exists j < i : x \dot{\mapsto} \Sigma_j \in D_{i-1}\} = \emptyset$.*

Proof. If $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab}$ is consistent, then clearly the two conditions hold. For the inverse direction, $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ and $D'_{i,Lab}$ are consistent. There are two ways inconsistencies can arise. There can be some decision d such that $d \in D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ and $\neg d \in D'_{i,Lab}$. Or the inconsistency occurs when forming the closure of $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab}$. The former case can only happen if there is an $x \in V$ such that $x \dot{\mapsto} \Sigma_j \in D_{i-1}$ for some $j < i$. The latter case occurs only, when $x \dot{\not\mapsto} \Sigma_i \in D'_{i,Lab}$ and $x \dot{\mapsto} \Sigma_i \in Clo(D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab}) \setminus (D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab})$. This happens, when $x \doteq y \in D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq}$ and $y \dot{\mapsto} \Sigma_i \in D'_{i,Lab}$. Thus there is a y such that $y \in V$ but $[y]_\Pi \not\subseteq V$. ■

Correctness and Completeness

We presume the correctness and completeness of the original decomposition with basic optimisations, as stated in Proposition 4.2.3.

Correctness

Lemma 4.4.5 *For each i with $1 \leq i \leq n$, the decision set $D_i|_{\mathcal{U}_i}$ is closed, consistent and complete for component i.*

Proof. $D_i|_{\mathcal{U}_i}$ is consistent as a subset of the consistent set D_i. $D_i|_{\mathcal{U}_i}$ is closed, because it is the reduction of the closed set D_i that contains all decisions over variables $\mathcal{U}_i$.

Let $x, y \in \mathcal{U}_i$. Then either $x \doteq y \in D_i|_{\mathcal{U}_i}$ or $x \not\doteq y \in D_i|_{\mathcal{U}_i}$ due to Step 1 of the algorithm. And either $x \dot{\mapsto} \Sigma_i \in D_i|_{\mathcal{U}_i}$ or $x \dot{\not\mapsto} \Sigma_i \in D_i|_{\mathcal{U}_i}$ due to Step 2. If $x \dot{\mapsto} \Sigma_i, y \dot{\not\mapsto} \Sigma_i \in D_i|_{\mathcal{U}_i}$ then immediately by Step 3 either $x \dot{<} y \in D_i|_{\mathcal{U}_i}$ or $y \dot{<} x \in D_i|_{\mathcal{U}_i}$. Therefore $D_i|_{\mathcal{U}_i}$ is complete for component i by Definition 4.4.1. ■

Proposition 4.4.6 *If for all i with $1 \leq i \leq n$ there exists a Σ_i-substitution σ_i that solves $(\Gamma_i, D_i|_{\mathcal{U}_i})$, then the input problem Γ is solvable.*

Proof. D_n is consistent by definition. Define the following generalised linear constant restriction $L = (\Pi, Lab, <_L)$ by

- $x \equiv_\Pi y$, iff $x \doteq y \in D_n$,
- $Lab(x) = \begin{cases} \Sigma_i, & \text{if } x \dot{\mapsto} \Sigma_i \in D_n, \\ \Sigma_n, & \text{otherwise;} \end{cases}$
- $<_L$ is given by any consistent extension of
  $x <_L y$, if $x \dot{<} y \in D_n$
  that orders each two variables with different labels.

L satisfies D_n, and if σ_i solves $(\Gamma_i, D_i|_{\mathcal{U}_i})$ then σ_i solves (Γ_i, L). Thus the input problem Γ is solvable due to correctness of the original algorithm (Proposition 4.2.3). ■

Completeness

The aim is to show the following

Proposition 4.4.7 *If the input problem Γ is solvable, then Γ is solvable by iterative decomposition.*

We will prove this proposition using the completeness of the original algorithm with basic optimisations. Due to the completeness of the original algorithm, if the input problem is solvable, there exists a generalised linear constant restriction L such that the output tuples $((\Gamma_i, L))_{1 \leq i \leq n}$ are solvable. This generalised linear constant restriction is used to guide the choices that will be made in each iteration of the iterative method.

Definition 4.4.8 Let $L = (\Pi, <_L, Lab)$ be a generalised linear constant restriction. Define

the set of equality decisions
$D_{\downarrow=} := \{x \doteq y \mid x \equiv_\Pi y \text{ and } \exists i \le n : x, y \in \mathcal{U}_i\}$,
the set of disequality decisions
$D_{\downarrow\neq} := \{x \dot{\neq} y \mid x \not\equiv_\Pi y \text{ and } \exists i \le n : x, y \in \mathcal{U}_i\}$,
the set of labelling decisions
$D_{\downarrow Lab} := \{x \dot{\mapsto} \Sigma_i \mid Lab(x) = \Sigma_i \text{ and } x \in \mathcal{U}_i\}$,
the set of ordering decisions as the set
$D_{\downarrow<} := \{x \dot{<} y \mid x <_L y, \exists j : x, y \in \mathcal{U}_j, (Lab(x) = \Sigma_j,$
$Lab(y) \neq \Sigma_j) \text{ or } (Lab(x) \neq \Sigma_j, Lab(y) = \Sigma_j)\}$.

Set $D_{\downarrow L}$, the *decision set induced by* L, as the closure of the union $D_{\downarrow=} \cup D_{\downarrow\neq} \cup D_{\downarrow<} \cup D_{\downarrow Lab}$.

Lemma 4.4.9 *$D_{\downarrow L}$ is a closed consistent set.*

Lemma 4.4.10 *Let Γ_i be a constraint problem. Let $L = (\Pi, Lab, <_L)$ be a linear constant restriction and $D_{\downarrow L}$ the decision set induced thereby. Then (Γ_i, L) is solvable , if and only if $(\Gamma_i, D_{\downarrow L}|_{\mathcal{U}_i})$ is solvable, where $D_{\downarrow L}|_{\mathcal{U}_i}$ is $D_{\downarrow L}$ restricted to decisions over variables $\mathcal{U}_i$.*

Proof. If (Γ_i, L) is solvable, then $(\Gamma_i, D_{\downarrow L}|_{\mathcal{U}_i})$ is solvable, because the decision set $D_{\downarrow L}|_{\mathcal{U}_i}$ induced by L contains only a subset of the decisions of L.

For the inverse direction, suppose σ solves $(\Gamma_i, D_{\downarrow L}|_{\mathcal{U}_i})$. If for $x, y \in \mathcal{U}_i : \sigma(x) = \sigma(y)$, then $x \doteq y \in D_{\downarrow L}|_{\mathcal{U}_i}$ and therefore $x \equiv_\Pi y$.
Now let $x \equiv_\Pi y$. Then $x \doteq y \in D_{\downarrow L}|_{\mathcal{U}_i}$ by definition of $D_{\downarrow L}$ and therefore $\sigma(x) = \sigma(y)$.

Let for $y \in \mathcal{U}_i : Lab(y) = \Sigma_j$ with $j \neq i$. If $y \in \mathcal{U}_j$, then $y \dot{\mapsto} \Sigma_j \in D_{\downarrow L}|_{\mathcal{U}_i}$. If $y \notin \mathcal{U}_j$, then there is no k such that $y \dot{\mapsto} \Sigma_k \in D_{\downarrow L}|_{\mathcal{U}_i}$. In both cases $y \dot{\mapsto} \Sigma_i \notin D_{\downarrow L}|_{\mathcal{U}_i}$. Therefore $\sigma(x) \in X$ as demanded.

Let for $x, y \in \mathcal{U}_i : Lab(x) = \Sigma_j, Lab(y) = \Sigma_i, j \neq i$ and $\sigma(x) \in \mathrm{Stab}(\sigma(y))$. Then $y \dot{\mapsto} \Sigma_i \in D_{\downarrow L}|_{\mathcal{U}_i}$; and $x \dot{\not\mapsto} \Sigma_i \in D_{\downarrow L}|_{\mathcal{U}_i}$ according to the same argument as in the previous paragraph. Therefore $x \dot{<} y \in D_{\downarrow L}|_{\mathcal{U}_i}$ and $x <_{L|_{\mathcal{U}_i}} y$ by definition of $D_{\downarrow L}|_{\mathcal{U}_i}$. ∎

We now have to show that $D_{\downarrow L}$ is a potential decision set calculated by the iterative decomposition.

Lemma 4.4.11 *Let (Γ, L) be a solvable component problem with decision set L. Then the induced decision set $D_{\downarrow L}$ can be constructed by the iterative decomposition, i.e., $D_{\downarrow L} = D_n$.*

Proof. In each component i, we make the following choices. Two variables $x, y \in \mathcal{U}_i$ are identified according to $D_{\downarrow L}$, that is, iff $x \doteq y \in D_{\downarrow L}$, then $x \doteq y \in D'_{i,=}$; iff $x \dot{\neq} y \in D_{\downarrow L}$, then $x \dot{\neq} y \in D'_{i,\neq}$. Iff $x \dot{\mapsto} \Sigma_i \in D_{\downarrow L}$, then $x \dot{\mapsto} \Sigma_i \in D'_{i,Lab}$. Iff $x \dot{<} y \in D_<$, then $x \dot{<} y \in D'_{i,<}$.

Claim 1: For $0 \leq i \leq n$: D_i is consistent and $D_i \subseteq D_{\downarrow L}$.

Proof of Claim 1:
$D_0 = \emptyset$ is obviously consistent and a subset of $D_{\downarrow L}$.
Let $i > 0$. $D_{i-1} \subseteq D_{\downarrow L}$ by hypothesis. $D'_{i,=}, D'_{i,\neq}, D'_{i,Lab}$ and $D'_{i,<}$ are subsets of $D_{\downarrow L}$ by definition, thus $D_{i-1} \cup D'_{i,=} \cup D'_{i,\neq} \cup D'_{i,Lab} \cup D'_{i,<}$ is consistent, because it is a subset of the consistent set $D_{\downarrow L}$. D_i defined as the closure of the above union is a subset of $D_{\downarrow L}$ by monotonicity of the closure operator and consistent, because it is a subset of a consistent set.

Claim 2: $D_{\downarrow L} = D_n$.

Proof of Claim 2:
$D_n \subseteq D_{\downarrow L}$ by Claim 1.
Let $x \doteq y \in D_{\downarrow L}$, then $x \doteq y \in Clo(D_{\downarrow =})$. $D_{\downarrow =} = \bigcup_{i=1}^n D'_{i,=}$ by definition, thus $Clo(D_{\downarrow =}) = Clo(\bigcup_{i=1}^n D'_{i,=}) \subseteq D_n$.
Let $x \not\doteq y \in D_{\downarrow L}$. Then, by definition, $x \not\doteq y \in Clo(D_{\downarrow =} \cup D_{\downarrow \neq})$. $D_{\downarrow =} \subseteq D_n$ by the above. If $w \not\doteq z \in D_{\downarrow \neq}$, then there is a j such that $w, z \in \mathcal{U}_j$, and thus $w \not\doteq z \in D'_{j,\neq}$. Therefore $D_{\downarrow \neq} \subseteq D_n$. Thus $x \not\doteq y \in D_n$, since D_n is closed.
Let $x \mathrel{\dot{\mapsto}} \Sigma_i \in D_{\downarrow L}$ for some i. Then $x \in \mathcal{U}_i$ by definition, and therefore $x \mathrel{\dot{\mapsto}} \Sigma_i \in D'_{i,Lab} \subseteq D_n$.
Concerning the ordering, $D_{\downarrow <} = \bigcup_{i=1}^n D'_{i,<}$ by definition. $Clo(D_{\downarrow =} \cup D_{\downarrow \neq}) = Clo(\bigcup_{i=1}^n D'_{i,=} \cup \bigcup_{i=1}^n D'_{i,\neq})$ by the above. Now $x \mathrel{\dot{<}} y \in D_{\downarrow L}$ implies $x \mathrel{\dot{<}} y \in Clo(D_{\downarrow <} \cup D_{\downarrow =} \cup D_{\downarrow \neq}) = Clo(\bigcup_{i=1}^n D'_{i,<} \cup \bigcup_{i=1}^n D'_{i,=} \cup \bigcup_{i=1}^n D'_{i,\neq}) \subseteq D_n$.

Claim 3: All of the above choices of the sets $D'_{i,=}, D'_{i,\neq}, D'_{i,Lab}, D'_{i,<}$ are valid as steps in iterative decomposition.

Proof of Claim 3:
That all of these choices can be made consistently, is shown by Claim 1.
For variable identification, the partitioning is directly given by the equivalence classes that result when restricting the equality and disequality decisions of $D_{\downarrow L}$ to the variables of a particular component problem, as done in $D'_{i,=}$ and $D'_{i,\neq}$.
For labelling, the generalised linear constant restrictions ensure that each variable receives only one label and that classes of variables that are identified receive one and the same label.
For ordering, $D'_{i,<}$ contains only ordering decisions on variables of component i. It respects the variable identification, because the generalised linear constant restriction $(\Pi, <_L, Lab)$ does so. And, by definition, each pair $x, y \in \mathcal{U}_i$ of variables where one has component i as label while the other has not is ordered in $D'_{i,<}$. ∎

Proof of Proposition 4.4.7.
Let Γ be solvable. By Proposition 4.2.3 there exists a generalised linear constant restriction $L = (\Pi, <_L, Lab)$ such that the output tuples $((\Gamma_i, L))_{1 \leq i \leq n}$ have a solution. By Lemma 4.4.10 the output tuples $((\Gamma_i, D_{\downarrow L}|_{\mathcal{U}_i}))_{1 \leq i \leq n}$ with the induced decision set $D_{\downarrow L}$ are solvable. By Lemma 4.4.11, there exists a set of choices of the iterative algorithm such that the decision set $D_{\downarrow L}$ is constructed thereby. ∎

4.5 An Algorithm for Computing the Variable Orderings

The description for choosing linear orderings in the iterative algorithm lays out only the conditions that an actual implementation has to fulfil. It is not a method for computing them. In this section we want to give a concrete algorithm, the one that is actually used in the implementation. The key point of the difficulty is the best use of the already given partial order that one has to respect when constructing the current one. One could generate all possible orderings and use the given partial order as a filter. But that would naturally lead to many orderings constructed in vain. The optimal solution would be one that integrates the filter into the construction and produces only such orderings that are compatible with the partial ordering. Of course we also want to respect the basic optimisation that two adjacent variables with identical label (here: current component or not current component) need not be ordered relative to each other. All of these demands are fulfilled in the following algorithm.

Function Restrictions

```
Input: - partial order PO from previously handle components
         given as a list, each element: a pair, a node and a set of
         successors which contains all immediate successors and maybe
         some more
         a node can can be marked as "done"
         (this is much simpler than actually taking the node out of
          the PO.)
       - list of variables of the current component VL
         (only combination variables are relevant)
         information whether variable is handled as a true variable or
         constant is given (by indexing)
         variables can be marked, markings are
         * "inserted" : variable is already inserted into the linear
           order
         * "blocking" : variable is inserted and blocks other
           variables in the PO above itself from being inserted later

Construction of one order

Let LO be the linear order to be constructed,
Set LO := []  (empty list)
Let VO be the list of variables still to insert
Set VO := VL  (without markings)
Let V/C-flag be a flag indicating whether the last block constructed
 was a block of variables or constants.
Non-deterministically set V/C-flag to V or C.

While VO =\= [] do
   {
   (Construction of the next block)
   Let BE-flag be a flag stating whether the current block is still
   empty
   Set BE-flag := true
```

```
   Invert V/C-flag  (V <--> C)
   If there are only variables of type V/C-flag left in VO
      Append VO to LO and exit the loop

   (Step through VO from left to right)
   For each element X in VO
      {
      If X is the last element of type V/C-flag in VO
       and BE-flag = true,
          take X
      else
          non-deterministically choose X
      Let SG be the set of elements below X in PO
      Set SG := Suborder(X, PO, VL)
      If SG =\= [] do
         {
         Set BE-flag := false
         Delete SG from VO
         Append SG to LO
         Mark X in VL as "blocking"
         Mark all other elements form SG in VL as "inserted"
         Mark all elements from SG as "done" in PO
         }
      }
   Delete all marks from VL;
   }

Remove all marks in PO.
Compute the linear constant restrictions from LO and call the
Constraint solving algorithm of the current component problem.
If the solver was successful,
   Integrate LO in PO
else
   backtrack choosing a new LO.
```

Function Suborder

```
Input:  Variable X
        Partial order PO with mark signs
        List of variables of the current component
         with markings VL
Output: If X can be integrated into the current linear order, the
        suborder below X in PO, else the empty list.

X can be integrated into the current linear order, if for all
variables Y below X in PO holds:
  Y is not in the current component (in VL)
or
  Y is of the same type (variable or constant) as X and not marked as
  "blocking".
The returned suborder is a list of variables that still have to be
appended to the quasi-linear order under construction.
```

```
If X is not in PO (a new variable, not handled so far)
   exit the function with return value [X].

Let Stack be a stack of elements from PO still to be processed
Set Stack := [X]
Let CL be a list of elements which don't need to be processed any
more.
Set CL := [].
Let SG be the suborder to be returned
Set SG := [].

While Stack =\= [] do
   {
   Set Y := Top(Stack)
   Set Stack := Pop(Stack)
   (Examine Y)
   If Y is marked as "done" in PO
      just continue
   If Y has a label different from X
   or Y is marked as "blocking" in VL
      set SG := [], Stack := []
   If Y is not in VL  (not in current component)
   or Y is marked as "inserted" in VL
      {
      Set CL := CL (set-) union Successors of Y (in PO)
      Set Stack := Push(Successors of Y minus CL, Stack)
      }
   else (Y in current component, not inserted, unmarked)
      {
      Set CL := CL union Successors of Y
      Set Stack := Push(Successors of Y minus CL, Stack)
      Set SG := [Y|SG]  (append Y to SG)
      }
   }

Return SG.
```

4.6 The Deductive Method

The method to describe in this section relies on the fact that many decisions in the search space are not really non-deterministic, but rather determined by demands of the components. It has been developed by J. Richts and is explained in full detail in his doctoral dissertation. We present it here, because any description of optimisation techniques would be incomplete without it, and also because the method has been developed in a co-operative project with us. The deductive and the iterative method are integrated to form one system.

A severe disadvantage of the original combination algorithm is that all non-deterministic decisions are made blindfolded without respecting the requirements that the components may impose. For example, if a component is an equational theory E_i that is collapse-free and the problem contains an equation $x = f(\dots y \dots)$ where $f \in \Sigma_i$, then x must receive label Σ_i. If E_i is also regular then the problem is unsolvable if $y \mathrel{\dot{\not\mapsto}} \Sigma_i \in D$ and $x \mathrel{\dot{<}} y \in D$. Hence the algorithm can choose $x \mathrel{\dot{\mapsto}} \Sigma_i \in D$ deterministically and take into account that $y \mathrel{\dot{\not\mapsto}} \Sigma_i \in D$ implies $y \mathrel{\dot{<}} x \in D$.

As the example shows, some decisions that have been deduced earlier in one component can be used to deduce new decisions in another component. This possible interplay between different components suggests to use a method where component algorithms computing new decisions are called alternately in the beginning of the combination algorithm and whenever a non-deterministic choice has been made: Starting with some initial decisions, each component algorithm computes new decisions; these new decisions are added to the current set of decisions, which is used when calling the other component algorithms. When this process comes to an end because no new decisions can be deduced, the next non-deterministic choice has to be made by the combination algorithm. After this choice the process of computing new consequences can be started again. At any step of computing the consequences, a component algorithm may return the information that its subproblem has become unsolvable with the current set of decisions. Thereby, unsolvable branches of the search tree can be detected earlier.

Obviously, this method requires new component algorithms that are capable of computing consequences implied by the component structure, the problem, and the decisions computed so far. A quasi-free structure for which such an algorithm does not exist can still be used in this method, but it cannot contribute to the deductive process. It is clearly the quality of the deductive component algorithms that decides the amount of optimisation achieved. The optimisations of our component algorithms go quite beyond using only syntactic properties of theories as in the example above. The goal is to deduce as much information as is possible with a reasonable effort.

The Algorithm

First we define the task of the new deductive component algorithms. Their input is a pure constraint problem and a set of decisions which need not be complete. The result is a set of decisions that follows from the constraint problem and the input decisions. If the input is unsolvable, the result may also be an inconsistent set of decisions.

Definition 4.6.1 Let (Γ, D) be a constraint problem with decision set. The decision set D is a *consequence* of (Γ, D), iff D is contained in every complete decision set $D' \supseteq D$ with (Γ, D') is solvable, i.e.,

$$C \subseteq \bigcap \{D' \mid D \subseteq D', D' \text{ is complete for } \Gamma, \text{ and } (\Gamma, D') \text{ is solvable } \}.$$

Note that $C = \emptyset$ is always a consequence and that the solution need not be inconsistent if (Γ, D') is unsolvable for all complete extensions D' of D. Therefore, the standard algorithms for constraint problems with linear constant restrictions must be called in the end when a complete set of decisions is reached. In the subsection on component algorithms, we discuss how deductive and component algorithms co-operate.

Now we can describe the algorithm. The termination condition in case of success is that the set of decisions is complete, as given in Lemma 4.3.7. In the following, D denotes the current set of decisions, initialised with $D := \emptyset$.

Repeat
    Deduce consequences:
    Repeat
        For each component i,
            call the component algorithm of component i to calculate
            new consequences D of (Γ_i, D),
            set the new current set of decisions $D := D \cup C$
    Until D is inconsistent
      or no component algorithm computes new decisions.
    If D is consistent and not complete
        Select next choice:
        Select a decision $d \notin D$ such that $D \cup \{d\}$ is consistent.
        Non-deterministically choose either
        $D := D \cup \{d\}$ or
        $D := D \cup \{\neg d\}$
Until D is inconsistent or complete
Return D.

Like the algorithms presented so far, this algorithm non-deterministically computes a decision set D for which each $(\Gamma_i, D|_{\mathcal{U}_i})$ has to be tested for solvability.

Proposition 4.6.2 *The input problem Γ is solvable, iff the algorithm computes a consistent decisions set D such that for each $i = 1, \ldots, n$ the constraint problem with decision set $(\Gamma_i, D|_{\mathcal{U}_i})$ is solvable.*

A proof of this proposition can be found in Richts' doctoral dissertation.

Deterministic Combination

It is interesting to observe that there exists a class of constraint systems for which the deductive combination algorithm has *PTIME* complexity, which entails that all steps can be made deterministically. In [94, 96], K. U. Schulz gives a general description of a *PTIME* combination algorithm for certain equational theories. This algorithm is extended to the combination of quasi-free structures in Chapter 6. The class of structures that are deterministically combinable is quite restricted. Currently, only unitary regular collapse-free structures are known to belong to it.

Although our deductive component algorithm is designed for the general case, it turns out to be an implementation of the deterministic algorithm when applied to component algorithms satisfying the conditions imposed in [94, 96] and Chapter 6. Our component algorithms for unification in the empty theory, for rational tree algebras, and for feature structures meet these conditions. Thus, when applied to these structures, our combination algorithm runs deterministically. This deterministic behaviour shows the great impact of interchanging decisions between component algorithms.

Component Algorithms

In order to prune the search space significantly, new component algorithms are needed for the deductive method. When designing these algorithms one should take into account the special way in which they are called. Many constraint solving algorithms, especially standard unification algorithms, are "one shot" algorithms: They are started only once with all information they need given and compute final results. Deductive component algorithms must be able to cope with partial information and deliver a meaningful but not not necessarily the final result. More importantly, when receiving new information the algorithms should not restart computation from scratch but rather continue on the base of their prior internal states. Otherwise, the search space would be partially shifted from the combination algorithm to the deductive component algorithms. The same holds for the standard component algorithms for constraint problems with linear constant restrictions that perform a complete test at the end of the combination algorithm: They should take into account the information already computed by the corresponding deductive component algorithms. Thus there should be a strong coupling of the standard and the deductive algorithm.

Note that there is no need for completeness in the deductive component algorithm. The algorithm need not compute all decisions implied by the input and it need not return an inconsistent set if the problem is unsolvable. Thus an algorithm returning always the empty set would be correct. This would not result in any optimisation, but it enables us to use every quasi-free structure in the deductive combination algorithm for which an algorithm for solving constraint problems with generalised linear constant restrictions exists. In the other extreme it might not be advisable to compute new decisions at any cost; there should be a careful consideration between optimisations of the combination algorithm resulting from new decisions and a higher complexity of the deductive component algorithm.

We developed deductive component algorithms for particular equational theories and equational unification: For the free theory, A, AC, and ACI. This is not the place to give a detailed description of these algorithms. In the following, we rather outline the ideas underlying them.

The deductive algorithm for the free theory is based on computing the most general unifier (mgu). Identification, labelling, and ordering can easily be computed from this mgu. The mgu has to be computed only once, namely when the deductive algorithm is called for the first time. When the algorithm is called with some new identification $x \doteq y$, which was deduced by another deductive algorithm, unification of the terms $\mathrm{mgu}(x)$ and $\mathrm{mgu}(y)$ has to be performed. All other decisions do not trigger any computation. This method can be integrated in the quasi-linear algorithm described in [17] where terms and unifiers are represented as directed acyclic graphs.

The theory $A = \{x+(y+z) = (x+y)+z\}$, i.e., the theory of an associative function symbol $+$ is basically the theory of free word equations. The deductive component algorithm translates the input into word equations and simplifies them. The simplification steps allow the computation of new identification, labelling and ordering information. This is an example of a deductive component algorithm which does not compute all consequences. Hence we need to call the standard algorithm for A-unification with linear constant restrictions in the end.

For the theory $AC = \{x+(y+z) = (x+y)+z; x+y = y+x\}$, i.e., the theory of an associative and commutative function symbol $+$, the deductive algorithm is based on [107]. First, the set of minimal solutions of the homogeneous Diophantine equations corresponding to the unification problem is computed. Some of these solutions can be deleted with the help of the existing decisions. From the remaining set of solutions, information about labelling, ordering and identification can be deduced.

The set of minimal solutions has to be recomputed when new identification decisions occur. This might seem to be a drawback at first glance, since computing the solutions of Diophantine equations can be a time-consuming task; but it cannot be worse than in the original combination algorithm, i.e., Diophantine equations are not solved more often, since this happens at most once for every partition of variables. Unfortunately, the number of minimal solutions of the Diophantine equations can be exponential in the size of the unification problem. But at least we do not need to compute complete sets of unifiers, which can even be doubly-exponential in number.

In the theory of Abelian monoids, $ACI = AC \cup \{x+x = x\}$, the binary function symbol is associative, commutative and idempotent. In [59] an algorithm was given that decides solvability of ACI-unification with constants. The main idea is to set up Horn clauses which describe the solvability of the equations. We extended this idea by defining more general Horn clauses such that variables can be turned into constants during the algorithm without changing the form of the Horn clauses. The algorithm works by propagating truth values through the clauses signalling insolvability when a contradiction occurs. New decisions can be deduced from the literals in the Horn clauses labelled with truth values during this propagation. Again, the Horn clauses must be set up from scratch, when new identification decisions occur.

The algorithms for the free theory and for the theories AC and ACI have in common that they behave like decision procedures for unification with linear constant restrictions if called with a complete set of decisions, i.e., they return a correct and complete answer. Therefore the final test does not need to compute anything; it can simply return the result achieved by the corresponding deductive component algorithm.

Rational Trees and Feature Structures

As examples of a quasi-free structures which are not an equational theories the author implemented rational tree algebras and feature structures of the Smolka and Treinen variety [104]. The algorithm for rational tree algebras is a simple extension of the algorithm for syntactic unification. The occurs-check has to be left out and the computation of new decisions is a bit more complicated since certain cyclic solution which are impossible in the free theory have to be taken into account.

We introduced feature structures as examples of quasi-free structures in 3.2.17. The implementation employs techniques for integrating record like data types (as feature structures) into logic programming frameworks developed by Van Roy, Mehl and Scheidhauer [114]. Upon first call, the internal graph-like representation of the feature theory is constructed and used to calculate new identification, labelling and ordering information. This representation needs to be constructed only once. Later on, new incoming identification information does not trigger a complete new setup, rather starts a feature structure unification of the the two structures pending below the newly identified variables. Additional information can be read out of the new structures, if unification succeeds. Incoming labelling or ordering information triggers no unification. Labelling information can help to deduce more information on the ordering. The algorithm is designed in such a way that it behaves like a decision procedure for feature constraint problems with linear constant restrictions when called with a complete set of decisions.

4.7 Integrating the Deductive and Iterative Method

The two methods described above can easily be integrated. The iterative method is a selection strategy for non-deterministic steps, while the deductive method deduces deterministic consequences from the decisions already made. Therefore integration is achieved by plugging the iterative selection strategy into the deductive algorithm. The combined method looks as follows. Suppose component constraint problems Γ_1 to Γ_{i-1} are solved, the current decision set is D, and D is not complete for component i, the current component. Select a decision $d \notin D$ over the variables of component i such that $D \cup \{d\}$ is consistent. Nondeterministically choose d or its negation and add it to D. Compute consequences and add them to D. If D is still not complete for component i, select the next decision for this component. If D is complete and $(\Gamma_i, D|_{\mathcal{U}_i})$ is solvable, proceed to the next component problem. Otherwise perform backtracking and make an alternative choice for one of the decisions made so far.

The method to compute consequences of a non-deterministic decision should be amended to the new selection strategy as follows. Components that are already solved cannot contribute any new decisions. Consequently only components that still have non-deterministic choices left open are consulted.

Tests

The above described optimisation methods and component algorithms have been implemented in COMMON LISP using the KEIM toolkit [53]. Indeed, there exists an implementation for the following component algorithms: for the free theory, the theories A, AC, and ACI and also for rational tree algebras and feature constraints. Several versions of the combintion algorithms representing different levels of optimisation are also implemented to test the specific contributions that the individual methods provide. Hence there is an implementation of the original combination algorithm with basic optimisations, two versions of the iterative method, two of the deductive method and two of the integrated solution. In the following we show some results of our optimisations. In order to test our algorithms with examples that occur in practice we used the REVEAL theorem prover [25]. For some example theorems, we collected all unification problems that are generated and solved by REVEAL while proving them. These theorems (and the corresponding sets of unification problems) contain free function symbols and constants and one or two AC-symbols.

Table 4.1 gives an overview of the run time for some sets of unification problems. The first six lines contain all unification problems that have to be solved by REVEAL during the proof search or completion of the respective example. The first three examples are very simple completions or proofs and the next three are more complex theorems from the REVEAL distribution. All examples except the first one contain two AC-symbols and several free symbols. The last three examples, containing several AC and ACI symbols, are added to demonstrate the potential of the iterative method. An empty cell in the columns indicates that the algorithm was aborted after running one hour.

We want to emphasise the differences between column 'ded' and 'ded-'. Column 'ded-' shows the run time of the algorithm when using only syntactic properties as described in [14]; a comparison with column 'ded' demonstrates the power of the deductive method and the deductive component algorithms. The run time decreases dramatically for most examples and some examples even cannot be solved in suitable time when using only syntactic properties.

The first six examples present an unexpected deficit in performance increase when using the iterative selection strategy in the deductive method (compare columns 'i+d' and 'ded'). This requires an explanation. Note that the number of backtracking steps is about the same. The equations in the example sets contain at most two AC-function symbols besides the free symbols. And our deductive component algorithms for AC and for the free theory are very sophisticated; they are capable of deducing such an amount of decisions that the remaining search space is to small to be shrinkable by the iterative selection strategy. The last three examples show that the use of the iterative selection strategy can lead to a speed-up by more than one order of magnitude. The equa-

		Time in seconds							Bktrk	
Example	Size	i+d	ded	i+d-	ded-	it+	it	orig	i+d	ded
Abelian group	29	3.7	3.7	5.0	5.0	3.2	11.6	17.2	4	4
Boolean ring	51	3.2	3.2	4.8	4.8	2.7	3.5	3.3	0	0
Boolean algebra	122	15.8	15.7	20.5	24.5	807			12	12
`exboolston`	87	12	12	948	997				17	14
`exgrobner`	1002	154	155	1442	1488				65	66
`exuqsl2`	404	109	108						74	74
AC^*–ACI^* 1	1	16	101	74	385	8.2	15		16	103
AC^*–ACI^* 2	1	31	407	393		413	841		13	205
AC^*–ACI^* 3	1	67	557			204	248		22	192

Legend: Bktrk: Number of Backtracking Steps; i+d: Integration of Iterative and Deductive Method; ded: Deductive Method; i+d-: Iterative & Deductive Method, but AC-component replaced by one that uses only collapse-freeness and regularity; ded-: Deductive Method, but AC-component replaced by one that uses only collapse-freeness and regularity; it+: Iterative Method alone, plus using collapse-freeness and regularity once at the start; it: Iterative Method alone; orig: Original unoptimised algorithm.

Table 4.1: Run time of some example sets

tions in these examples contain several AC and ACI-function symbols besides free function symbols. It is a general observation that the iterative method is advantageous, if the number of components is large or the deductive component algorithms do not deduce many decisions.

In order to get more examples, we developed a test set generator. With it, one generates sets of random combined unification problems over signatures containing several function symbols from different theories. Certain means were taken to ensure that about half of the generated problems are solvable. Table 4.7 presents some run time results for these randomly generated problem sets. The signature contains 2 A, 2 AC, 0–3 ACI and several free function symbols. The problems are that complex that a use of a combination method different from the deductive combination makes no sense at all.

It is interesting to observe that with these problems, the iterative selection strategy is not always the best choice. There are examples (sets 2, 3, and 15) in which the iterative selection strategy is superior. On the other hand, in the sets 1, 5, 6, and 18 it is much worse than a strategy which firstly settles all variable identification and discrimination decisions for all component problems. It is currently not clear what the conditions are under which one should choose the iterative selection strategy, and when to rather use the other strategy. The presence of several collapsing theories is important, but there are several collapsing theories both in those examples where the iterative selection strategy works well and in those where it flounders. In all these examples, it seems important to make the "right" decisions first, but there is at current no way to state what the "right" decisions are.

Set	Equations	term-depth	# *ACI*	Ded+Iter time	Ded+Iter bktrk	Ded time	Ded bktrk
1	199/98	6	3	816	1953	81	152
2	200/99	6	3	232	780	>1h	
3	199/101	6	3	330	800	1158	1982
4	200/127	6	3	58	250	42	110
5	200/97	6	3	1362	3971	141	401
6	200/113	6	3	>1h		103	295
7	200/112	6	3	676	2217	189	689
8	200/100	5	0	19	1	19	1
9	200/90	5	0	67	33	75	33
10	200/95	5	0	16	1	15	1
11	200/87	5	0	20	7	21	10
12	200/89	5	0	21	8	21	8
13	200/99	5	1	32	50	31	30
14	200/93	5	1	21	47	26	22
15	200/109	5	1	154	394	3931	12335
16	200/116	5	1	26	50	30	31
17	200/107	5	2	319	1116	83	147
18	200/106	5	2	1250	2627	44	107
19	200/95	5	2	178	462	58	169
20	200/108	5	2	99	414	43	159

Legend: The signature of these problems consists of 2 A, 2 AC, 0–3 ACI and several free function symbols. Equations: number of equations in set and number of solvable equations; term depth: maximal depth of terms; # ACI: Number of ACI-function symbols in signature; Ded+Iter: deductive combination with iterative selection strategy; Ded: deductive combination with a selection strategy that chooses all identifications first; bktrk: number of backtracking steps.

Table 4.2: Run time of randomly generated example sets

Another observation is that there is no simple, e.g., linear, connetion between the run time and the number of backtracking steps. Obviously, some backtracking steps require a lot of time, because they appear high up in the search tree, while others that are close to the leaf nodes of the search tree have a very small influence on the run time.

4.8 Related Work

We presented an optimised algorithm for deciding combined constraint problems on the basis of the iterative and the deductive method. The test section indicates that the optimisations deliver an impressive speed-up over the practically unusable naive implementation. But for most equational theories E, the complexity of general E-unification is NP-hard (see [94, 95]). Since general E-unification can be regarded as a combination of E-unification with constants and syntactic unification, it therefore follows, that no algorithm how optimised it may be will ever be capable of solving all unification problems in polynomial time, unless $NP = P$. This means though we see the practical usability of our algorithm, we do not claim that all problems are solved.

The work that is most closely related to ours is the one by Boudet [20, 21], where an efficient method for combining equational unification algorithms that compute complete sets of unifiers is presented. The most important difference is that our algorithm is designed to handle combined decision problems for constraint systems while Boudet's computes complete sets of unifiers for equational unification problems. Consequently, the theories combinable by Boudet's method must be finitary equational theories, i.e., the minimal complete set of unifiers always has to be finite. This is not the case for decision procedures. For equational theories, this may not make such a big difference, because most theories used in theorem proving are finitary. An exception is the theory A of an associative function symbol, which is infinitary, but decidable. The decision procedure given by Makanin [71] has EXPSPACE complexity (see [48]) which is quite bad, but, e.g., no worse than computing complete sets of AC-unifiers. Hence it could well become usable for certain applications.

If one progresses from unification theory to constraint solving, it is of course important that one is not forced to calculate and apply sets of unifiers. Quasi-free structures such as rational tree algebras and feature structures are efficiently decidable, but unifiers cannot be computed for them due to their nature. Consequently, Boudet's method cannot be used here. We, on the other hand, implemented deductive components for these structures to show their practical usability.

For the restriction to finitary equational theories, Boudet describes a method to simulate the algorithm by Baader and Schulz within his framework. But unfortunately under these conditions, he looses all the optimisations he had earlier introduced. His simulation is indeed a naive implementation of the decision procedure by Baader and Schulz. And there is no way to enhance the situation with his optimisation techniques, because they rely on information resulting from the computed sets of unifiers.

Our algorithm contains three sources of non-determinism, namely the variable identification, the labelling and the ordering. Boudet's algorithm possesses an equivalent for each of the three sources of non-determinism. Variable identification is exactly matched, it occurs in the E_i-resolution step. Labelling and ordering occur, too, but in a somewhat different setting. Boudet's labelling rule, called **Mark**, is applied only after a theory conflict is discovered. In contrast, we always label all variables. But Boudet's labelling is superior only at first sight. What is the reason for the need of a labelling? A conflict in which two theories simultaneously try to instantiate one and the same variable. If both theories are collapse-free, the situation cannot be remedied, the input problem has no solution. Both algorithms have special optimisations to handle collapse-free theories. A resolution of the theory conflict is possible only, if one theory has collapsing axioms. In that case, it is a matter of chance, if Boudet's algorithm is superior. If, by coincidence, the unifiers that are selected in the resolution steps in Boudet's algorithm happen to be compatible, i.e., the collapse axiom is used to avoid the theory conflict, Boudet needs no labelling. If coincidentally a different pair of unifiers is chosen, the theory conflict arises, and an application of his **Mark** rule is required. Our algorithm does not make a blind shot to see if it hits by coincidence a non-conflicting pair of unifiers, but labels straight away. This is obviously not much worse.

Boudet's method of handling the ordering is advantageous. His algorithm guesses a particular ordering of a pair of variables only in case where the solution computed so far contains a (compound) cycle. In our method, it depends on the component algorithms. If the component algorithms are good, they may be able to determine the order of a pair of variables. Otherwise we have to guess the order non-deterministically.

Boudet's algorithm contains a further source of non-determinism that is totally absent in our algorithm: The choice of a particular unifier in the E_i-resolution step. If the task is to compute all unifiers for a given combined unification problem, there is no other way, one has to compute complete sets of unifiers in all E_i-resolution steps. But Boudet also aims at deciding unification problems with his method. In this case the selection of a particular unifier out of the complete set of unifiers for a single component resolution step introduces a source of non-determinism, that is not required.

It is questionable that picking a unifier out of a complete set of unifiers is a promising strategy. There is little doubt about its correctness. But the non-determinism contained seems to be inappropriately handled. Especially the arbitrary selecting contains a mixture of don't-know and don't-care non-determinism. What does that mean? A single unifier contains all the information that is special to this particular solution of the problem as well as the general information that is common to all solutions, because the problem cannot be solved elsehow. Suppose a unifier identifies two variables. Unfortunately one cannot determine whether this identification is proprietary to this unifier or shared by all unifiers in the complete set. But it makes an important difference. If the identification is special for this unifier, it is a don't-know non-determinism, otherwise it is a don't-care non-determinism. Suppose solving the overall problem fails because of this identification. In the don't-care case, no backtracking is required; since all unifiers share this identification, any unifier is as bad as the one chosen. In the don't-know case, the situation is different. Then one must backtrack to find a unifier that avoids the failure causing identification. Thus one should rather know whether the choice one makes is a don't-know or a don't-care choice. It is this problem that the deductive method tries to tackle. It is the task of the deductive components to find out as much as possible about the "don't-care" part, the part that must be present in any solution independent of what particular solution one finally picks. So, by deducing what part is common to all solutions, one can avoid a lot of unnecessary backtracking.

There is still another problem in using sets of solutions. As stated, Boudet does not just test a component for solvability, he calculates a unifier and uses that unifier. The unifier is interpreted as a solved form of the component problem. So in a sense, it is a component problem, but a trivially solved one. Now, there are equational theories, for which the solved form of a unification problem can be exponentially large in the size of the input problem. An example of such a theory is AC, as can be seen in [38]. Thus after a resolution step, Boudet's algorithm continues computation with a component that may have grown exponentially in size. The point is, there is no guarantee that each component is solved only once. Later resolutions of different components, especially certain identifications of shared variables, may cause the earlier solved component to be no longer in solved form. Hence it has to be solved again, but now it may be – due to earlier resolution – exponentially larger. And the resolution step may again lead to an exponential growth in size. This is the reason why currently the worst case complexity of Boudet's algorithm is unknown. As seen by J. Richts and A. Boudet himself,[2] it seems possible that the complexity is k-EXPTIME for some $k \geq 2$ or even non-elementary. On the other hand, the decision problem is in NP and a naive implementation of the unoptimised algorithm by Baader and Schulz is singly-exponential, as shown in Subsection 4.2.

We conclude with a positive observation. An important aspect of the iterative method is the localisation of choices. This strategy is very much present in Boudet's algorithm. The variables considered for identification in an E_i-resolution step are the ones of the component constraint problem to be currently resolved. Variables subject to the **Mark** rule are the ones that caused a clash, so they are certainly local to the problem and the components involved. And variables subject to the **Cycle**-rule are variables in a compound cycle. Again these are local to the components involved in the cycle. One could argue that the localisation in Boudet's method is driven to the utmost point, which is a strength of his method.

4.9 Conclusion

We presented optimisation techniques for combining constraint solvers in this chapter. Starting from the original algorithm and its basic optimisations, we described two orthogonal methods to achieve the task of a noteworthy optimisation. The iterative method is a strategy for selecting non-deterministic decisions. It is based on the insight that component constraint problems should be solved one by one. Otherwise one faces the danger of testing in vain sets of linear constant restrictions that differ in choices for some component when some other is plain unsolvable. We showed that non-deterministic decisions can be made locally for one component without loosing soundness or completeness, if these decisions are adequately propagated to the other components. The deductive method provides a way to deduce the consequences of a certain non-deterministic decision made. One can observe quite often that once

[2] Personal communications.

a decision is made certain components require that other choices have to be made in a particular way for their subproblems to remain solvable. Thus after each non-deterministic decision the components are consulted to state what consequences they require. This method, developed using techniques of constraint propagation, proves enormously effective in shrinking the search space, provided specialised component algorithms that efficiently compute large sets of consequences are available. To this end, the development of good deductive component algorithms is as important as the deductive combination method itself. The orthogonality of the iterative and the deductive method allow for a simple integration into a common setup. We think this integrated algorithm is a good choice for practical applications of combining constraint solvers. This view is supported by our test results. To do even better would probably include the step away from general combination methods and solution domains that we presented here to very particular ones that make sense only for the special component constraint solvers and solution domains given in a particular situation.

Chapter 5

Rational Amalgamation

5.1 Introduction

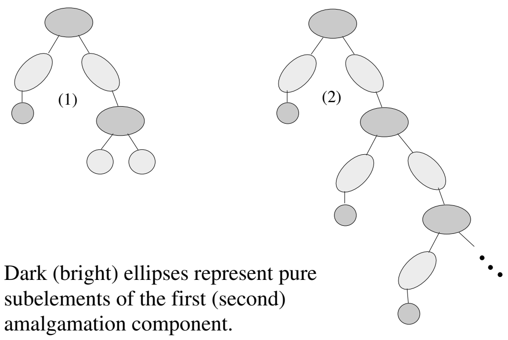
A general combination method, in our sense, has to give answers to two problems. First, it must offer a general construction for combining two solution domains. Second, a combination algorithm has to be given that reduces the problem of solving "mixed" constraints over the combined solution domain to the problem of solving "pure" constraints over the two component structures. In Chapter 3, we described a first such general method of combination developed by F. Baader and K. U. Schulz [10, 12, 15]: The free amalgamated product. It is characterised as being the most general combined solution domain of all structures that can be reasonably considered as combinations of two components. For quasi-free structures over disjoint signatures, an explicit construction of the free amalgamated product of two components is given, and an algorithm is presented that combines the constraint solvers of the components to gain a constraint solver for the free amalgam.

In this chapter, we introduce a second systematic way to combine constraint systems over quasi-free structures, called *rational amalgamation*. Free and rational amalgamation both yield a combined structure with "mixed" elements that interweave a finite number of "pure" elements of the two components in a particular way. The difference between both constructions becomes transparent when we ignore the interior structure of these pure subelements and consider them as construction units with a fixed arity, similar to "complex function symbols". Under this perspective, and ignoring details, mixed elements of the free amalgam can be considered as finite trees, whereas mixed elements of the rational amalgam are like rational trees[1]. The following picture gives an impression of this view.

On this background it should not be surprising that in praxis rational amalgamation appears to be the preferred combination principle in situations where the two solution structures to be combined are themselves "rational" or "cyclic"

[1]A possibly infinite tree is *rational* if it is finitely branching and has only a finite number of distinct subtrees. See [29, 36, 70].

Mixed element of free amalgam (1) and of rational amalgam (2).



domains: for example, it represents the way how rational trees and rational lists are interwoven in the solution domain of Prolog III [30], and a variant of rational amalgamation has been used to combine feature structures with non-wellfounded sets in a system introduced by W. Rounds [90].

We introduce rational amalgamation as a general construction that can be used to combine so-called non-collapsing quasi-free structures over disjoint signatures. The elements of the amalgam can in fact be regarded as rational trees where each node is labelled with an element of one component. It is then shown how constraint solving in the rational amalgam can be reduced to constraint solving in the components. The decomposition scheme that is used is closely related to the decomposition algorithm for free amalgamation, but it avoids one non-deterministic step that is needed in the latter scheme. Hence, when matters of efficiency become important, rational amalgamation might be the better choice.

Let us now briefly indicate which insights could be gained from a classification of basic methodologies for combining constraints systems. Below we shall summarise what has been obtained so far.

1. It helps to understand the scale of possibilities and the general limitations for combining constraints systems.

2. It might facilitate the design of new combined constraint systems, and it helps to understand existing instances of combination from a general point of view.

3. It establishes new and interesting connections between the theory of constraint solving and other areas such as, e.g., universal algebra and logic.

4. The relationship between different methodologies for combining constraint systems is interesting per se, we hope to verify.

1. From our present perspective, which is explained in more detail in Section 5.6, free and rational amalgamation, and a related construction called

"infinite amalgamation" seem to be the most important combination principles in a spectrum of related methods. Furthermore, we are confident that the abstract definition of a quasi-free structure, as introduced in [10] and used here, captures a maximal class of (unsorted!) structures where these combination principles can be applied in a uniform way. This class covers most of the non-numerical and non-finite solution domains that are used in constraint solving. All the solution domains that are considered in the area of unification modulo equational theories are quasi-free structures. Furthermore, the algebra of rational trees, feature structures, and structures with finite or rational nested sets, lists and multi sets are quasi-free structures.

2. The results presented in this chapter show, e.g., that there is a common and general methodology behind Colmerauer's combination of rational trees and rational lists in the solution domain of Prolog III [30] and Rounds' combination of feature structures with non-wellfounded sets [90]. The amalgamation technique to be described in this chapter can be used, e.g., to obtain similar combinations where rational trees, feature structures, rational lists, nested multi sets, or quotient term algebras for collapse-free equational theories over disjoint signatures are interweaved in arbitrary manner.

3. The purely algebraic definition of a quasi-free structure directly generalises the notion of a free structure (see Section 3.2.2 and [15] for a thorough discussion). Still, quasi-free structures have what is sometimes called the "universal mapping property" of free structures, and a major part of the theory of free structures as developed in universal algebra can be lifted to the case of quasi-free structures. A detailed mathematical investigation of this point is in progress. Furthermore, it has turned out that the methods for combining solution domains developed in [10] and here, and the general methods for combining logics described by Gabbay [44] and Pfalzgraf [81, 82] follow the same abstract idea. See [44] for a first discussion of this issue.

4. One interesting connection between free and rational amalgamation is the observation that the free amalgamated product is always a substructure of the rational amalgamated product. Section 5.6 will be used to comment on item 4 in more detail.

We would like to point out that the theoretical concepts and the mathematical methods underlying "braids" and their simplification were developed in cooperation with K. U. Schulz.

5.2 Non-collapsing Quasi-free Structures

In this section we shall introduce the class of structures for which we can use the rational amalgamation construction (Definition 5.2.5). It will be a subclass of quasi-free structures, which are introduced and thoroughly discussed in Section 3.2.2. Still, we repeat here a few of the notions introduced in that section to help the reader remember the context. The algebra of rational trees will be used to exemplify the concepts. In the sequel, we consider a fixed Σ-structure $\mathfrak{A}^\Sigma$, and $\mathcal{M}$ denotes a submonoid of $End_{\mathfrak{A}}^\Sigma$.

The stable hull (see Definition 3.2.9) of a set A_0 has properties that are similar to those of the subalgebra generated by A_0: $\mathrm{SH}_{\mathcal{M}}^{\mathfrak{A}}(A_0)$ is always a Σ-substructure of $\mathfrak{A}^\Sigma$, and $A_0 \subseteq \mathrm{SH}_{\mathcal{M}}^{\mathfrak{A}}(A_0)$. In general, however, the stable hull can be larger than the generated subalgebra. For example, if $\mathfrak{A}^\Sigma := R(\Sigma, X)$ denotes the algebra of rational trees over signature Σ, if $\mathcal{M} = End_{\mathfrak{A}}^\Sigma$, and if $Y \subseteq X$ is a subset of the set of variables, X, then $\mathrm{SH}_{\mathcal{M}}^{\mathfrak{A}}(Y)$ contains all *rational* trees with variables in Y, while Y generates all *finite* trees with variables in Y only.

The set $X \subseteq A$ is an *$\mathcal{M}$-atom set for* $\mathfrak{A}^\Sigma$ if every mapping $X \to A$ can be extended to an endomorphism in $\mathcal{M}$. If $\mathcal{M} = End_{\mathfrak{A}}^\Sigma$, then X is simply called an *atom set* for $\mathfrak{A}^\Sigma$. For example, if $\mathfrak{A}^\Sigma := \mathfrak{R}(\Sigma, X)$ is the algebra of rational trees over the set of variables X, then X is an atom set for $\mathfrak{A}^\Sigma$. Remember that the extension of every mapping $X \to A$ to an endomorphism of $\mathfrak{A}^\Sigma$ in $\mathcal{M}$ is *unique* (Lemma 3.2.19).

A countably infinite Σ-structure $\mathfrak{A}^\Sigma$ is a quasi-free structure, iff there exists a submonoid $\mathcal{M}$ of $End_{\mathfrak{A}}^\Sigma$ such that $\mathfrak{A}^\Sigma$ has an infinite $\mathcal{M}$-atom set X where every element $a \in A$ is stabilised by a finite subset of X with respect to $\mathcal{M}$. (Definition 3.2.14).

Examples 3.2.17 contain a long list of examples of quasi-free structures. Amongst them, one finds free structures, vector spaces, rational tree algebras, hereditarily finite well-founded and non-wellfounded sets and lists, and certain types of feature structures.

In the rest of this section, $(\mathfrak{A}^\Sigma, X, \mathcal{M})$ denotes a fixed quasi-free structure with carrier A.

Lemma 5.2.1 *Let $\varphi(v_1, \ldots, v_k)$ be a positive Σ-formula, let $m \in \mathcal{M}$, and let $a_1, \ldots, a_k$ be elements of A. Then $\mathfrak{A}^\Sigma \models \varphi(v_1/a_1, \ldots, v_k/a_k)$ implies $\mathfrak{A}^\Sigma \models \varphi(v_1/m(a_1), \ldots, v_k/m(a_k))$.*

Proof. It is simple to see that there exists a surjective endomorphism $m' \in \mathcal{M}$ that coincides with m on $\{a_1, \ldots, a_k\}$. The result follows from the fact that validity of positive formulae is preserved under surjective homomorphisms (Lemma 2.1.1). ■

Lemma 5.2.2 *Let $\varphi(v_1, \ldots, v_k)$ be a positive Σ-formula, and let $x_1, \ldots, x_k$ be distinct atoms in X. Then $\mathfrak{A}^\Sigma \models \varphi(v_1/x_1, \ldots, v_k/x_k)$ implies $\mathfrak{A}^\Sigma \models \forall v_1, \ldots \forall v_k\ \varphi$.*

Proof. Let $a_1, \ldots, a_k$ be arbitrary elements of A. Since X is an $\mathcal{M}$-atom set, there exists an $m \in \mathcal{M}$ such that $m(x_i) = a_i$, for $i = 1, \ldots, k$. Hence $\mathfrak{A}^\Sigma \models \varphi(v_1/x_1, \ldots, v_k/x_k)$ implies $\mathfrak{A}^\Sigma \models \varphi(v_1/a_1, \ldots, v_k/a_k)$, by Lemma 5.2.1. It follows that $\mathfrak{A}^\Sigma \models \forall v_1 \ldots \forall v_k\ \varphi$. ■

For each element $a \in A$, there exists a unique minimal subset $Y \subset X$ of the atom set such that $\{a\} \in \mathrm{SH}^{\mathfrak{A}}_{\mathcal{M}}(Y)$, the stabiliser of a. For the mathematical treatment of quasi-free structures, the concept of the stabiliser turns out to be extremely useful. It might give a good intuition to imagine that the stabiliser of an element a is the set of atoms "occurring" in a. If a is an element of an algebra of rational trees over the set of variables V, then the stabiliser of a is in fact the set of variables occurring in the rational tree a. Note, however, that "the" set of atoms (variables) occurring, e.g., in distinct terms that represent the same element of a quotient term algebra is *not* unique in general. It is trivial to see that $\mathrm{SH}^{\mathfrak{A}}_{M}(Y) = \{a \in A \mid \mathrm{Stab}^{\mathfrak{A}}_{M}(a) \subseteq Y\}$, for each $Y \subseteq X$. In the sequel, further properties of stabilisers will be used. The first lemma is a trivial consequence of the fact that stable hulls are Σ-substructures.

Lemma 5.2.3 *Let $f \in \Sigma$ be an n-place operator and $a_1, \ldots, a_n \in A$. Then* $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(f_{\mathfrak{A}}(a_1, \ldots, a_n)) \subseteq \mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(\{a_1, \ldots a_n\})$.

The next lemma plays a crucial role in the rational amalgamation construction. It will be used in many proofs.

Lemma 5.2.4 *Let $m \in \mathcal{M}$ be an endomorphism of the quasi-free structure $(\mathfrak{A}^{\Sigma}, X, \mathcal{M})$ such that the restriction of m on X is a mapping $X \to X$. If* $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(a) = \{x_1, \ldots, x_k\}$, *then* $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(m(a)) \subseteq \{m(x_1), \ldots, m(x_k)\}$. *If m is an automorphism, then* $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(m(a)) = \{m(x_1), \ldots, m(x_k)\}$.

Proof. Let m_1 and m_2 be two endomorphisms in $\mathcal{M}$ that coincide on $\{m(x_1), \ldots, m(x_k)\} \subseteq X$. Then $m_1 \circ m$ and $m_2 \circ m$ are endomorphisms in $\mathcal{M}$ that coincide on $\{x_1, \ldots, x_k\}$. By assumption, $m_1 \circ m$ and $m_2 \circ m$ coincide on a. But then m_1 and m_2 coincide on $m(a)$. Hence $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(m(a)) \subseteq \{m(x_1), \ldots, m(x_k)\}$. Assume that m is an automorphism, and that $\mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(m(a))$ is a proper subset of $\{m(x_1), \ldots, m(x_k)\}$. The first part of the lemma, applied to m^{-1}, yields a proper subset of $\{x_1, \ldots, x_k\}$ that stabilises a, which is impossible, by choice of $\{x_1, \ldots, x_k\}$. ■

We may now characterise the subclass of quasi-free structures for which we can use the rational amalgamation construction.

Definition 5.2.5 A quasi-free structure $(\mathfrak{A}^{\Sigma}, X, \mathcal{M})$ is *non-collapsing* if every endomorphism $m \in \mathcal{M}$ maps non-atoms to non-atoms (i.e., $m(a) \in A \setminus X$ for all $a \in A \setminus X$ and all $m \in \mathcal{M}$).

For example, quotient term algebras for collapse-free equational theories, rational tree algebras, feature structures, feature structures with arity, the domains with nested, finite or rational lists, and the domains with nested, finite or rational multi sets (as mentioned in 3.2.17) are always non-collapsing.

Let us note that the domains with nested, finite or rational sets do not belong to the class of non-collapsing quasi-free structures. The reason is that in this case atoms have the form $\{y\}$, where y is taken from a countably infinite set of urelements Y. Since we do not use sorts, and since union of urelements is not defined, the urelements itself do not belong to the structure. If y_1 and y_2 are distinct urelements, then $\{y_1, y_2\} = \{y_1\} \cup \{y_2\}$ is a non-atomic element. Now any endomorphism that maps the atom $\{y_1\}$ to $\{y_2\}$ and leaves $\{y_2\}$ fixed, "collapses" the non-atom $\{y_1, y_2\}$ to the atom $\{y_2\}$.[2]

5.3 The Domain of the Rational Amalgam

In this section we shall define the underlying domain of the rational amalgam of two non-collapsing quasi-free structures over disjoint signatures. This is the most complicated step of the rational amalgamation construction. For this reason we start with a discussion that motivates the following abstract definitions.

As we indicated in the introduction, we would like to lift the usual construction of rational trees, where nodes are labelled with the function symbols of a fixed signature, to a higher level where we interweave elements of distinct structures. Unfortunately, the classical notion of a tree is not really a useful basis for realising this idea, as long as we do not want to impose severe restrictions on the two components. The reason is that, classically, trees are either ordered or unordered. None of these concepts seems appropriate to model a situation where we want to interweave, say, both ordinary terms (representing ordered trees) with nested sets (representing unordered trees) or with elements of arbitrary quotient term algebras (where the "tree status" is doubtful).

In this section we shall see that there exists a natural notion that captures the idea of a generalised tree built with the elements of two structures. We introduce the concept of a "braid" where the problem of the correct order between the successors of a node is completely abstracted away. Basically, the links from a parent "node" (an element of one component) to its successor nodes (a finite number of elements belonging to distinct components) are organised by set theoretical functions that connect suitable atoms of the parent node with its successor elements. In this way, the ordering of the links depends (only) on the way how atoms in the parent node are ordered.

One drawback of the concept of a braid is the fact that different braids may represent the same object. The major part of this section will be used to show that a nice standard normal form for each braid can be given. Then, the set of braids in standard normal form will represent the carrier of the rational amalgam.

In the sequel, we shall describe the rational amalgamation of two component structures. There are, however, no difficulties to interweave any finite number of components in the same way.

[2]We think that this unpleasant effect disappears when we use sorts. With sorts, it should be possible to use the set of urelements as atom set. Of course $\{y_1, y_2\}$ may still be mapped to $\{y_2\}$, but the latter is a non-atomic element now.

Throughout this section $(\mathfrak{A}^{\Sigma}, X, \mathcal{M})$ and $(\mathfrak{B}^{\Delta}, Y, \mathcal{N})$ denote two fixed non-collapsing quasi-free structures over disjoint signatures. We assume that the atom sets X and Y have the form $X = Z \uplus \mathcal{O}_A$ and $Y = Z \uplus \mathcal{O}_B$, where the sets $Z, \mathcal{O}_A$, and $\mathcal{O}_B$ are all infinite, and where $\mathcal{O}_A \cap \mathcal{O}_B = \emptyset$. The atoms in Z will be called *bottom atoms*, the atoms in $\mathcal{O}_A$ ($\mathcal{O}_B$) will be called *open atoms*. In the braid construction, the bottom atoms will play the role of ordinary atoms, or leaves. Open atoms, in contrast, can be considered as "named holes" that are only used to link elements of both structures. With $\mathcal{O}_A(a)$ and $\mathcal{O}_A(A')$ we denote the set of open atoms occurring in the stabiliser of $a \in A$ ($A' \subseteq A$) with respect to $\mathcal{M}$. Similarly expressions $\mathcal{O}_B(b)$ ($\mathcal{O}_B(B')$) are used to denote the set of open atoms occurring in the stabiliser of $b \in B$ ($B' \subseteq B$) with respect to $\mathcal{N}$.

An endomorphism $m \in \mathcal{M}$ ($n \in \mathcal{N}$) is called *admissible* if m (n) leaves all bottom atoms $z \in Z$ fixed and if $m(o) \in \mathcal{O}_A$ ($n(o) \in \mathcal{O}_B$) for all $o \in \mathcal{O}_A$ ($o \in \mathcal{O}_B$).[3] Automorphisms are called admissible if they define a permutation of the set of open atoms while leaving bottom atoms fixed. A pair $(m, n) \in \mathcal{M} \times \mathcal{N}$ is called admissible if both m and n are admissible.

Lemma 5.3.1 *Let $A' \subseteq A$. If the admissible endomorphisms $m_1, m_2 \in \mathcal{M}$ coincide on $\mathcal{O}_A(A')$, then m_1 and m_2 coincide on A'. Similarly, let $B' \subseteq B$. If the admissible endomorphisms $n_1, n_2 \in \mathcal{N}$ coincide on $\mathcal{O}_B(B')$, then n_1 and n_2 coincide on B'.*

5.3.1 Braids and Subbraids

Before we introduce braids, let us formalise the type of links that we shall use to interweave elements of two components.

Definition 5.3.2 Let $\mathcal{O}'_A \subseteq \mathcal{O}_A$, $\mathcal{O}'_B \subseteq \mathcal{O}_B$, let $\pi_A : \mathcal{O}'_A \to B$, $\pi_B : \mathcal{O}'_B \to A$, let $\pi := \pi_A \cup \pi_B$. An element $a \in A$ is *directly linked to* $b \in B$ via π if there is an $o \in \mathcal{O}_B(b)$ such that $a = \pi_B(o)$. Analogously $b \in B$ is directly linked to $a \in A$ via π if there exists an $o \in \mathcal{O}_A(a)$ such that $b = \pi_A(o)$. An element $a \in A \cup B$ is a π*-descendant of* $b \in A \cup B$ if there exists a sequence $a = a_0, a_1, \ldots, a_n = b$ ($n \geq 0$) such that each a_i is directly linked to a_{i+1} via π, for $0 \leq i \leq n-1$.

Definition 5.3.3 A *braid* of type A over $\mathfrak{A}^{\Sigma}, \mathfrak{B}^{\Delta}$ is a quintuple $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$, where

1. $a \in A \setminus \mathcal{O}_A$,
2. C is a finite subset of A containing a. All elements of $C \setminus \{a\}$ are *non-atomic*. D is a finite set of *non-atomic* elements of B,
3. $\pi_A : \mathcal{O}_A(C) \to D$ and $\pi_B : \mathcal{O}_B(D) \to C$ are mappings. For $\langle o, e \rangle \in \pi_A \cup \pi_B$, e is always a *non-atomic* element,

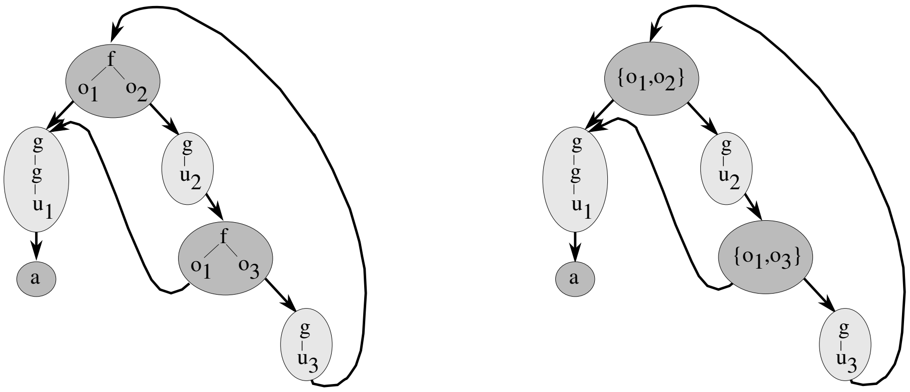
[3] Intuitively, admissible endomorphisms cause a "renaming" of open atoms, compare Lemma 5.2.4. They may identify distinct open atoms.

4. each element in $C \cup D$ is a π-descendant of a, for $\pi := \pi_A \cup \pi_B$.

The element a is called the *root* of $\mathcal{K}$. The elements in the sets C and D are called the *elements of* $\mathcal{K}$ *of type* A *and* B respectively. The functions π_A and π_B are called the *linking functions of* $\mathcal{K}$ *of type* A *and* B respectively.

Braids of type B, with root in $B \setminus \mathcal{O}_B$, are defined symmetrically. A braid $\mathcal{K}$ is called *trivial* if the root of $\mathcal{K}$ is a bottom atom $z \in Z$. In this case, z is the only element of the braid. It does not make sense to distinguish between the trivial braid $\langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$ of type A and the trivial braid $\langle z, \emptyset, \{z\}, \emptyset, \emptyset\rangle$ of type B. We identify both braids. Hence, trivial braids have mixed type.

Example 5.3.4 The graph on the left-hand side of the following figure represents a braid over two free term algebras, for signatures $\Sigma = \{f, a\}$ and $\Delta = \{g\}$ respectively. In this representation, the root is the topmost element $f(o_1, o_2)$. Note that this braid represents (modulo unfolding) an ordered tree, due to the fact that there exists a fixed order between the atoms occurring in the elements $f(o_1, o_2)$ and $f(o_1, o_3)$.



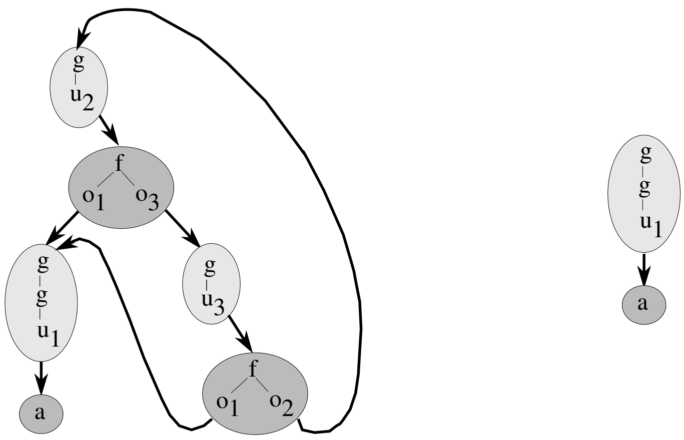
On the right-hand side, we interweaved in a similar way the elements $\{o_1, o_2\} = \{o_2, o_1\}$ and $\{o_1, o_3\} = \{o_3, o_1\}$ of an algebra with nested multi sets. Now the braid represents (modulo unfolding) an unordered tree.

We sometimes write $\mathcal{O}_A(\mathcal{K})$ and $\mathcal{O}_B(\mathcal{K})$ for $\mathcal{O}_A(C)$ and $\mathcal{O}_B(D)$ respectively, and $\mathcal{O}(\mathcal{K})$ denotes the union $\mathcal{O}_A(\mathcal{K}) \cup \mathcal{O}_B(\mathcal{K})$. A quintuple $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$ that satisfies Conditions 1-3 of Definition 5.3.3 will be called a *prebraid.*

Definition 5.3.5 Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$ be a braid. The braid $\mathcal{K}' := \langle a', C', B', \pi'_A, \pi'_B\rangle$ (of type A or B) is a *subbraid* of $\mathcal{K}$ if $a' \in C \cup D$, $C' \subseteq C$, $D' \subseteq D$, $\pi'_A \subseteq \pi_A$, and $\pi'_B \subseteq \pi_B$.

Sub(pre)braids of prebraids are defined in the same way. We write $\mathcal{K}' \subseteq \mathcal{K}$ if $\mathcal{K}'$ is a sub(pre)braid of $\mathcal{K}$.

Example 5.3.6 Here are two subbraids of the first braid given in Example 5.3.4. As above, the topmost element represents the root.



Lemma 5.3.7 *Let $\mathcal{K}_i = \langle a_i, C_i, D_i, \pi_A^i, \pi_B^i \rangle$ be a braid, and let $\pi^i = \pi_A^i \cup \pi_B^i$, for $i = 1, 2$. If $a_1 = a_2$, and if $\pi_1(o) = \pi_2(o)$ for all $o \in \mathcal{O}(\mathcal{K}_1) \cap \mathcal{O}(\mathcal{K}_2)$, then $\mathcal{K}_1 = \mathcal{K}_2$.*

Proof. A simple induction shows that each π_1-descendant of a_1 is a π_2-descendant of a_2, and vice versa. Hence both braids have the same elements. The second condition given in the lemma implies that both braids have the same linking functions. Hence $\mathcal{K}_1 = \mathcal{K}_2$. ∎

Corollary 5.3.8 *Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be two prebraids and $\mathcal{K}_1 \subseteq \mathcal{K}_2$. Let $\mathcal{K}_1'$ be a subbraid of $\mathcal{K}_1$ and let $\mathcal{K}_2'$ be a subbraid of $\mathcal{K}_2$ such that $\mathcal{K}_1'$ and $\mathcal{K}_2'$ have the same root. Then $\mathcal{K}_1' = \mathcal{K}_2'$.*

Proof. Since $\mathcal{K}_1'$ and $\mathcal{K}_2'$ are subbraids of $\mathcal{K}_2$ it is obvious that $\mathcal{K}_1'$ and $\mathcal{K}_2'$ satisfy the conditions of Lemma 5.3.7. Hence $\mathcal{K}_1' = \mathcal{K}_2'$. ∎

Lemma 5.3.9 *For each element e of a prebraid $\mathcal{K}$ there exists a unique subbraid of $\mathcal{K}$ with root e.*

Proof. Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ and $e \in C \cup D$. Then e cannot be an open atom. Let $C' \subseteq C$ ($D' \subseteq D$) be the set of π-descendants of e in C (resp. D), where $\pi = \pi_A \cup \pi_B$. Note that all elements of $(C' \cup D') \setminus \{e\}$ are non-atomic since $\mathcal{K}$ is a prebraid. Let $\pi_A' \subseteq \pi_A$ (resp. $\pi_B' \subseteq \pi_B$) contain all ordered pairs $\langle o, c \rangle$ of π_A (resp. π_B) where $o \in \mathcal{O}_A(C')$ (resp. $o \in \mathcal{O}_B(D')$). For each such pair $\langle o, c \rangle$ the element c is in D' (C') since c is a π-descendant of e. Since $\langle o, c \rangle \in \pi$, the element c is non-atomic. It follows that $\langle e, C', D', \pi_A', \pi_B' \rangle$ is a subbraid of $\mathcal{K}$. By Corollary 5.3.8 it is the unique subbraid of $\mathcal{K}$ with root e. ∎

5.3.2 Variants

The concrete open atoms that are used to organise links between elements of distinct type in a given braid should be regarded as irrelevant. This motivates the following definition.

Definition 5.3.10 Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ and $\mathcal{K}' = \langle a', C', D', \pi'_A, \pi'_B \rangle$ be two prebraids, say, of type A. $\mathcal{K}'$ is called a *variant* of $\mathcal{K}$ if there exists an admissible pair of automorphisms (m, n) such that

1. $a' = m(a)$,
2. $C' = \{m(c) \mid c \in C\}$, and $D' = \{n(d) \mid d \in D\}$,
3. $\pi'_A := \{\langle m(o), n(d) \rangle \mid \langle o, d \rangle \in \pi_A\}$, and
   $\pi'_B := \{\langle n(o), m(c) \rangle \mid \langle o, c \rangle \in \pi_B\}$.

Lemma 5.3.11 *If two prebraids are variants, then the two subbraids given by their roots are variants.*

Proof. Let $\mathcal{K}$ and $\mathcal{K}'$ be variants of the form as in the previous definition. Let $\mathcal{K}_1 = \langle a, C_1, D_1, \pi^1_A, \pi^1_B \rangle$ be the unique subbraid of $\mathcal{K}$ with root a, and let $\mathcal{K}_2 = \langle a, C_2, D_2, \pi^2_A, \pi^2_B \rangle$ be the unique subbraid of $\mathcal{K}'$ with root $a' = m(a)$. The elements in $C_1 \cup D_1$ are the π-descendants of a, for $\pi = \pi_A \cup \pi_B$. The elements in $C_2 \cup D_2$ are the π'-descendants of a', for $\pi' = \pi'_A \cup \pi'_B$. From the definition of π'_A and π'_B and from Lemma 5.2.4 (second part) it follows easily that the π'-descendants of $a' = m(a)$ are the m resp. n -images of the π-descendants of a. This shows that $C_2 = \{m(c) \mid c \in C_1\}$ and $D_2 = \{n(d) \mid d \in D_1\}$. The rest is obvious. ■

The following lemma shows that the notion of a variant gives rise to an equivalence relation on the set of all (pre)braids. Since the set of admissible automorphisms of $\mathfrak{A}^{\Sigma}$ (resp. $\mathfrak{B}^{\Delta}$) defines (with composition) a group, the proof is obvious.

Lemma 5.3.12 *Each prebraid $\mathcal{K}_1$ is a variant of $\mathcal{K}_1$. If $\mathcal{K}_2$ is a variant of the prebraid $\mathcal{K}_1$, then $\mathcal{K}_1$ is a variant of $\mathcal{K}_2$. If $\mathcal{K}_2$ is a variant of the prebraid $\mathcal{K}_1$, and if $\mathcal{K}_3$ is a variant of $\mathcal{K}_2$, then $\mathcal{K}_3$ is a variant of $\mathcal{K}_1$.*

Lemma 5.3.13 *Let (m, n) be an admissible pair of automorphisms. Let $\mathcal{K}$, a', C', D', π'_A, and π'_B be defined as in Definition 5.3.10, 1.-3. Then $\mathcal{K}' := \langle a', C', D', \pi'_A, \pi'_B \rangle$ is a prebraid and a variant of $\mathcal{K}$.*

Proof. Since $a \in A \setminus \mathcal{O}_A$ it follows that $a' = m(a) \in A \setminus \mathcal{O}_A$, by choice of m. Thus $\mathcal{K}'$ satisfies Condition 1 of Definition 5.3.3. Since m and n are admissible automorphisms, all elements of $C' \cup D'$ that are images of non-atomic elements of $C \cup D$ under m and n respectively are non-atomic. Hence

$\mathcal{K}'$ satisfies Condition 2 of Definition 5.3.3. Since m and n define permutations of $\mathcal{O}_A$ and $\mathcal{O}_B$ respectively, the first component o of each pair $\langle o, e\rangle$ in $\pi'_A \cup \pi'_B$ is an open atom. Lemma 5.2.4 shows that $o \in \mathcal{O}_A(C') \cup \mathcal{O}_B(D')$. Obviously, if $o \in \mathcal{O}_A(C')$, then $e \in D'$ and if $o \in \mathcal{O}_A(D')$, then $e \in C'$. Moreover, e is always non-atomic, by admissibility of m and n. Since m and n are automorphisms, π'_A and π'_B are functions. By Lemma 5.2.4, the domains of π'_A and π'_B are $\mathcal{O}_A(C')$ and $\mathcal{O}_B(D')$ respectively, which shows that $\mathcal{K}'$ satisfies Condition 3 of Definition 5.3.3. Thus $\mathcal{K}'$ is a prebraid. Clearly it is a variant of $\mathcal{K}$. ■

Lemma 5.3.14 *Each variant of a braid is a braid.*

Proof. It suffices to verify that each variant of a braid satisfies Condition 4 of Definition 5.3.3. Let $\mathcal{K}$, $\mathcal{K}'$, and (m, n) as in Definition 5.3.10. Suppose that $d \in D$ is directly linked to $c \in C$ via π_A. Thus, for some $o \in \mathcal{O}_A(c)$ we have $\langle o, d\rangle \in \pi_A$. Lemma 5.2.4 shows that $m(o) \in \mathcal{O}_A(m(c))$. Clearly $m(c) \in C'$. Since $\langle m(o), n(d)\rangle \in \pi'_A$, the element $n(d) \in D'$ is directly linked to $m(c) \in C'$. Now a simple induction shows that all elements of $C' \cup D'$ are π'-descendants of the new root $a' = m(a)$, where $\pi' = \pi'_A \cup \pi'_B$. ■

5.3.3 Simplification of Braids

Two (pre)braids that are variants of each other are meant to denote the same object. But then we should not distinguish between two subbraids of one and the same (pre)braid if they are variants. In order to identify such subbraids, we shall use admissible pairs of endomorphisms of a particular type.

Definition 5.3.15 The admissible pair of endomorphisms (m, n) is a *simplifier* for the prebraid $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$ if the following conditions hold:

- $\forall o_1, o_2 \in \mathcal{O}_A(C)$: $m(o_1) = m(o_2)$ implies $n(\pi_A(o_1)) = n(\pi_A(o_2))$,
- $\forall o_1, o_2 \in \mathcal{O}_B(D)$: $n(o_1) = n(o_2)$ implies $m(\pi_B(o_1)) = m(\pi_B(o_2))$.

Lemma 5.3.16 *Let (m, n) be a simplifier for the prebraid $\mathcal{K}$. Then (m, n) is a simplifier for each subprebraid of $\mathcal{K}$.*

Proof. Let $\mathcal{K}' = \langle a', C', D', \pi'_A, \pi'_B\rangle$ be a subprebraid of $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$. Then $C' \subseteq C$ and $D' \subseteq D$. Hence $\mathcal{O}_A(C') \subseteq \mathcal{O}_A(C)$ and $\mathcal{O}_B(D') \subseteq \mathcal{O}_B(D)$. Moreover, the functions π_A and $\pi'_A \subseteq \pi_A$ (π_B and $\pi'_B \subseteq \pi_B$) coincide on $\mathcal{O}_A(C')$ (resp. $\mathcal{O}_B(D')$). The rest is obvious. ■

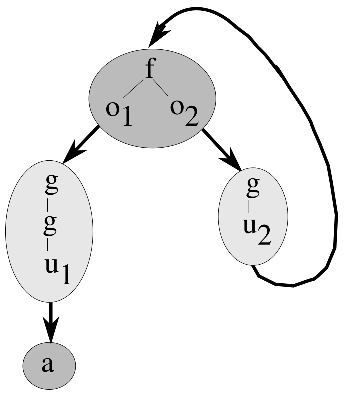
Definition 5.3.17 Let (m, n) be a simplifier for the prebraid $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$. The *image of $\mathcal{K}$ with respect to (m, n)* is the prebraid $\mathcal{K}^{(m,n)} := \langle a', C', D', \pi'_A, \pi'_B\rangle$ with the following components:[4]

[4]Using Lemma 5.2.4 and the fact that both $\mathfrak{A}^{\Sigma}$ and $\mathfrak{B}^{\Delta}$ are non-collapsing it is trivial to verify that $\mathcal{K}^{(m,n)}$ is a prebraid.

1. $a' := m(a)$,
2. $C' := \{m(c) \mid c \in C\}$ and
   $D' := \{n(d) \mid d \in D\}$,
3. $\pi'_A := \{\langle m(o), n(d)\rangle \mid \langle o, d\rangle \in \pi_A, m(o) \in \mathcal{O}_A(C')\}$, and
   $\pi'_B := \{\langle n(o), m(c)\rangle \mid \langle o, c\rangle \in \pi_B, n(o) \in \mathcal{O}_B(D')\}$.

Now assume that $\mathcal{K}$ is a braid. The *braid-image of* $\mathcal{K}$ *with respect to* (m,n), $\mathcal{K}^{\langle m,n\rangle}$, is the unique subbraid of $\mathcal{K}^{(m,n)}$ with root a'.

Example 5.3.18 The following figure represents the braid-image of the braid on the left-hand side in Example 5.3.4 under the simplification (m,n) where m maps o_3 to o_2 and n maps u_3 to u_2:



The next lemma gives a refinement of Lemma 5.3.16.

Lemma 5.3.19 *Let* (m,n) *be a simplifier for the prebraid* $\mathcal{K}$*, let* $\mathcal{K}_1$ *be the unique subbraid of* $\mathcal{K}$ *with root* e*, where* e *is an element of* $\mathcal{K}$ *of type* A *(resp.* B*). Then* $\mathcal{K}_1^{\langle m,n\rangle}$ *is the unique subbraid of* $\mathcal{K}^{(m,n)}$ *with root* $m(e)$ *(resp.* $n(e)$*).*

Proof. It follows directly from Definition 5.3.17 that $\mathcal{K}_1^{\langle m,n\rangle}$ is a subbraid of the prebraid $\mathcal{K}^{(m,n)}$. Obviously $m(e)$ (resp. $n(e)$) is the root of $\mathcal{K}_1^{\langle m,n\rangle}$. Now use Corollary 5.3.8. ∎

There is one technical point behind the definition of a simplifier that will cause some difficulties in the further development. Assume, in the situation of Definition 5.3.17, that $\mathcal{O}_A(C) = \{o_1, \ldots, o_k\}$ and $\mathcal{O}_B(D) = \{u_1, \ldots, u_l\}$. Then there is no guarantee that $\mathcal{O}_A(C') = \{m(o_1), \ldots, m(o_k)\}$ and $\mathcal{O}_B(D') = \{n(u_1), \ldots, n(u_l)\}$. In fact, Lemma 5.2.4 only shows the inclusion $\mathcal{O}_A(C') \subseteq \{m(o_1), \ldots, m(o_k)\}$.

Definition 5.3.20 The set

$$(\{m(o) \mid o \in \mathcal{O}_A(C)\} \setminus \mathcal{O}_A(C')) \;\cup\; (\{n(o) \mid o \in \mathcal{O}_B(D)\} \setminus \mathcal{O}_B(D'))$$

is called the set of *pending atoms* of the simplification step leading from $\mathcal{K}$ to $\mathcal{K}^{(m,n)}$.

As we shall see, pending atoms complicate the treatment of simplification. In principle we could restrict the amalgamation construction to a class of structures for which we can replace the inclusion from Lemma 5.2.4 mentioned above by an equality. In this case pending atoms cannot occur, image and braid image always coincide, and we could dispense with prebraids at all. However, our motivation was to give a general construction. For this reason we shall not follow this line.

Lemma 5.3.21 *Assume, in the situation of Definition 5.3.17, that $o \in \mathcal{O}_A(C)$ and $m(o)$ is not a pending atom of the simplification step leading from $\mathcal{K}$ to $\mathcal{K}^{(m,n)}$. Then $\pi'_A(m(o)) = n(\pi_A(o))$.*

Proof. A trivial consequence of the definition of π'_A as given in 5.3.17. ∎

While we are mainly interested in simplification of braids, it turns out to be simpler to treat simplification of prebraids in advance.

Lemma 5.3.22 *Let (m_1, n_1) be a simplifier for the prebraid $\mathcal{K}_0$ and (m_2, n_2) be a simplifier for the prebraid $\mathcal{K}_1 = \mathcal{K}_0^{(m_1,n_1)}$. Assume that m_2 and n_2 do not identify any pending atom of the simplification step leading from $\mathcal{K}_0$ to $\mathcal{K}_1$ with another atom. Then $(m_2 \circ m_1, n_2 \circ n_1)$ is a simplifier for $\mathcal{K}_0$ and $\mathcal{K}_0^{(m_2 \circ m_1, n_2 \circ n_1)} = \mathcal{K}_1^{(m_2,n_2)}$.*

Proof. Let $\mathcal{K}_i = \langle a_i, C_i, D_i, \pi^i_A, \pi^i_B \rangle$, for $i = 0, 1$. We may assume that both are of type A. Let $(m, n) := (m_2 \circ m_1, n_2 \circ n_1)$. If $m(o) = m(o')$ for $o, o' \in \mathcal{O}_A(C_0)$, then either $m_1(o) = m_1(o')$, or $m_1(o) \neq m_1(o')$ and $m_2(m_1(o)) = m_2(m_1(o'))$. In the former case we know that $n_1(\pi^0_A(o)) = n_1(\pi^0_A(o'))$ since (m_1, n_1) is a simplifier for $\mathcal{K}_0$. Hence $n(\pi^0_A(o)) = n(\pi^0_A(o'))$. In the latter case, neither $m_1(o)$ nor $m_1(o')$ can be pending, by assumption. Hence $m_1(o)$ and $m_1(o')$ are in $\mathcal{O}_A(\mathcal{K}_1)$. By Lemma 5.3.21, $n_1(\pi^0_A(o)) = \pi^1_A(m_1(o))$ and $n_1(\pi^0_A(o')) = \pi^1_A(m_1(o'))$. Since $m_2(m_1(o)) = m_2(m_1(o'))$ and (m_2, n_2) is a simplifier for $\mathcal{K}_1$, this implies that

$$n_2(n_1(\pi^0_A(o))) = n_2(\pi^1_A(m_1(o))) = n_2(\pi^1_A(m_1(o'))) = n_2(n_1(\pi^0_A(o'))).$$

Hence in both cases $n(\pi^0_A(o)) = n(\pi^0_A(o'))$. Symmetrically it follows that $n(o) = n(o')$ implies $m(\pi^0_B(o)) = m(\pi^0_B(o'))$, for all $o, o' \in \mathcal{O}_A(D_0)$. Hence (m, n) is a simplifier for $\mathcal{K}_0$.

The prebraids $\mathcal{K}_0^{(m_2 \circ m_1, n_2 \circ n_2)}$ and $\mathcal{K}_1^{(m_2,n_2)}$ have the same root $m_2(m_1(a_0))$. It is trivial that they have the same elements. But then it follows easily that they have the same linking functions. ∎

Corollary 5.3.23 *Let (m_1, n_1) be a simplifier for the prebraid $\mathcal{K}_0$ and (m_2, n_2) be a simplifier for the prebraid $\mathcal{K}_1 = \mathcal{K}_0^{(m_1,n_1)}$. Then there exists a simplifier (m, n) for $\mathcal{K}_0$ such that $\mathcal{K}_0^{(m,n)} = \mathcal{K}_1^{(m_2,n_2)}$.*

Proof. It follows from Lemma 5.3.1 that there exists a simplifier (m_2', n_2') of $\mathcal{K}_1$ such that (m_2', n_2') does not identify any pending atom of the simplification step leading from $\mathcal{K}_0$ to $\mathcal{K}_1$ with another atom, and $\mathcal{K}_1^{(m_2,n_2)} = \mathcal{K}_1^{(m_2',n_2')}$. Let $(m,n) := (m_2' \circ m_1, n_2' \circ n_1)$. Then, by the previous lemma, $\mathcal{K}_0^{(m,n)} = \mathcal{K}_1^{(m_2',n_2')} = \mathcal{K}_1^{(m_2,n_2)}$. ∎

Let $\mathcal{K}$ be a prebraid. We have seen that a simplifier (m,n) that yields a permutation of $\mathcal{O}_A$ and $\mathcal{O}_B$ leads to the variant $\mathcal{K}^{(m,n)}$ of $\mathcal{K}$ (Lemma 5.3.13). The same is true under weaker assumptions. By Lemma 5.3.1, the image $\mathcal{K}^{(m,n)}$ is completely determined by the images of the elements in $\mathcal{O}_A(\mathcal{K})$ and $\mathcal{O}_B(\mathcal{K})$ under the endomorphisms m and n respectively. Hence we obtain

Lemma 5.3.24 *Let (m,n) be a simplifier for the prebraid $\mathcal{K}$. If the restrictions of m and n on $\mathcal{O}_A(\mathcal{K})$ and $\mathcal{O}_B(\mathcal{K})$ respectively are injective, then $\mathcal{K}^{(m,n)}$ is a variant of $\mathcal{K}$.*

Call a simplifier (m,n) for $\mathcal{K}$ *strict* if the restriction of m on $\mathcal{O}_A(\mathcal{K})$ or the restriction of n on $\mathcal{O}_B(\mathcal{K})$ is *not* injective. Lemma 5.2.4 shows that $|\mathcal{O}(\mathcal{K}^{(m,n)})| < |\mathcal{O}(\mathcal{K})|$ if (m,n) is strict. It follows that

Lemma 5.3.25 *$|\mathcal{O}(\mathcal{K})|$ gives an upper bound on the length of every sequence of strict simplifications for the prebraid $\mathcal{K}$.*

A prebraid $\mathcal{K}'$ is called *irreducible* if $\mathcal{K}'$ does not have a strict simplifier. We want to show that all irreducible prebraids that can be reached from a prebraid $\mathcal{K}$ by simplification are variants. For this purpose, the following lemma is needed that shows that simplification of prebraids is "locally confluent".

Lemma 5.3.26 *Let (m_1,n_1) and (m_2,n_2) be two simplifiers for the prebraid $\mathcal{K}_0$, let $\mathcal{K}_1$ and $\mathcal{K}_2$ be the images of $\mathcal{K}_0$ under (m_1,n_1) and (m_2,n_2) respectively. Then there exist a simplifier (m_3,n_3) for $\mathcal{K}_1$ and a simplifier (m_4,n_4) for $\mathcal{K}_2$ such that $\mathcal{K}_1^{(m_3,n_3)} = \mathcal{K}_2^{(m_4,n_4)}$.*

Proof. Let $\mathcal{K}_i = \langle a_i, C_i, D_i, \pi_A^i, \pi_B^i \rangle$, for $i = 0,1,2$. The endomorphisms m_1 and n_1 define equivalence relations $\sim_A^1$ and $\sim_B^1$ on $\mathcal{O}_A(C_0)$ and $\mathcal{O}_B(D_0)$ respectively, where elements are equivalent with respect to $\sim_A^1$ ($\sim_B^1$) iff they have the same image under m_1 (n_1). The endomorphisms m_2 and n_2 define similar equivalence relations $\sim_A^2$ and $\sim_B^2$ on $\mathcal{O}_A(C_0)$ and $\mathcal{O}_B(D_0)$ respectively. Let $\sim_A := \sim_A^1 \sqcup \sim_A^2 = (\sim_A^1 \cup \sim_A^2)^*$ denote smallest equivalence relation on $\mathcal{O}_A(\mathcal{K}_0)$ that extends $\sim_A^1$ and $\sim_A^2$. Similarly, let $\sim_B$ denote the smallest equivalence relation on $\mathcal{O}_B(\mathcal{K}_0)$ that extends $\sim_B^1$ and $\sim_B^2$. Choose a system of representants for $\sim_A$ and a similar system for $\sim_B$. We shall write $rep(o)$ for the representant of $[o]$ with respect to $\sim_A$ ($\sim_B$), for $o \in \mathcal{O}_A(\mathcal{K})$ ($o \in \mathcal{O}_B(\mathcal{K})$).

The elements of $\mathcal{O}_A(C_1)$ have the form $m_1(o_A)$ for $o_A \in \mathcal{O}_A(C_0)$, by Lemma 5.2.4. If, for $o_A, o_A' \in \mathcal{O}_A(C_0)$, $m_1(o_A) = m_1(o_A') \in \mathcal{O}_A(C_1)$, then $o_A \sim_A^1 o_A'$ and $rep(o_A) = rep(o_A')$. Thus

- the mapping $m_1(o_A) \mapsto rep(o_A)$ $(o_A \in \mathcal{O}_A(C_0))$ is welldefined. It can be extended to an admissible endomorphism $m_3 \in \mathcal{M}$. Similarly the mapping $n_1(o_B) \mapsto rep(o_B)$ $(o_B \in \mathcal{O}_B(D_0))$ is welldefined and can be extended to an admissible endomorphism $n_3 \in \mathcal{N}$.

Symmetrically we can show

- the mapping $m_2(o_A) \mapsto rep(o_A)$ $(o_A \in \mathcal{O}_A(C_0))$ is welldefined and can be extended to an admissible endomorphism $m_4 \in \mathcal{M}$, and the mapping $n_2(o_B) \mapsto rep(o_B)$ $(o_B \in \mathcal{O}_B(D_0))$ is welldefined and can be extended to an admissible endomorphism $n_4 \in \mathcal{N}$.

We have $(*)$

$$\begin{aligned} m_3(m_1(o_A)) &= rep(o_A) = & m_4(m_2(o_A)) & \qquad (o_A \in \mathcal{O}_A(C_0)) \\ n_3(n_1(o_B)) &= rep(o_B) = & n_4(n_2(o_B)) & \qquad (o_B \in \mathcal{O}_B(D_0)) \end{aligned}$$

and, by Lemma 5.3.1, $(**)$

$$\begin{aligned} m_3(m_1(c)) &= m_4(m_2(c)) & \qquad (c \in C_0) \\ n_3(n_1(d)) &= n_4(n_2(d)) & \qquad (d \in D_0). \end{aligned}$$

Clearly (m_3, n_3) and (m_4, n_4) are admissible. We shall now show that (m_3, n_3) is a simplifier for $\mathcal{K}_1$. Let $\langle o'_1, b'_1\rangle, \langle o'_2, b'_2\rangle \in \pi^1_A$ and suppose that $m_3(o'_1) = m_3(o'_2)$. We have to verify that $n_3(b'_1) = n_3(b'_2)$. For $i = 1, 2$, there exists $o_i \in \mathcal{O}_A(C_0)$ and $b_i := \pi^0_A(o_i) \in D_0$ such that $o'_i = m_1(o_i)$ and $b'_i = n_1(b_i)$. Since m_3 identifies o'_1 and o'_2 we know that $o_1 \sim_A o_2$. Thus there exists a sequence $o_1 = u_1, u_2 \ldots, u_k = o_2$ such that each pair $\langle u_i, u_{i+1}\rangle$ belongs either to $\sim^1_A$ or to $\sim^2_A$ $(1 \leq i < k)$. Let $d_i := \pi^0_A(u_i)$, for $i = 1, \ldots, k$. Thus $d_i \in D_0$ $(1 \leq i \leq k)$ and we have $b_1 = d_1$ and $b_2 = d_k$. Now (m_1, n_1) and (m_2, n_2) are simplifiers. Thus, if $\langle u_i, u_{i+1}\rangle \in \sim^1_A$, then $n_1(d_i) = n_1(d_{i+1})$, which implies $n_3(n_1(d_i)) = n_3(n_1(d_{i+1}))$, and if $\langle u_i, u_{i+1}\rangle \in \sim^2_A$, then $n_2(d_i) = n_2(d_{i+1})$, which implies $n_4(n_2(d_i)) = n_4(n_2(d_{i+1}))$ and, by $(**)$, $n_3(n_1(d_i)) = n_3(n_1(d_{i+1}))$. Therefore we obtain in fact

$$n_3(b'_1) = n_3(n_1(b_1)) = n_3(n_1(b_2)) = n_3(b'_2).$$

We have shown that (m_3, n_3) is a simplifier for $\mathcal{K}_1$. Symmetrically it follows that (m_4, n_4) is a simplifier for $\mathcal{K}_2$.

The two prebraids $\mathcal{K}_1^{(m_3,n_3)}$ and $\mathcal{K}_2^{(m_4,n_4)}$ have the same root $m_3(m_1(a_0))$, by $(**)$. It is trivial to see that $(**)$ also implies that $\{m_3(m_1(c)) \mid c \in C_0\}$ is the set of elements of type A of both prebraids, and $\{n_3(n_1(d)) \mid d \in D_0\}$ is the set of elements of type B of both prebraids. But then it follows easily that both prebraids have the same linking functions, which means that they are identical. ∎

Theorem 5.3.27 *Each sequence of iterated strict simplifications that starts from the prebraid $\mathcal{K}$ has length $\leq |\mathcal{O}(\mathcal{K})|$. If $\mathcal{K}'$ is an irreducible prebraid that is obtained from $\mathcal{K}$ by a sequence of simplifications, then there exists a simplifier (m,n) for $\mathcal{K}$ such that $\mathcal{K}^{(m,n)} = \mathcal{K}'$. If two irreducible prebraids $\mathcal{K}_1$ and $\mathcal{K}_2$ can be reached from $\mathcal{K}$ by sequences of simplifications, then $\mathcal{K}_1$ and $\mathcal{K}_2$ are variants.*

Proof. The first statement is Lemma 5.3.25. The second statement follows from Corollary 5.3.23 with a trivial induction. If $\mathcal{K}_1$ and $\mathcal{K}_2$ are two irreducible prebraids that are obtained from $\mathcal{K}$ by sequences of simplifications, then both prebraids can be obtained from $\mathcal{K}$ by a single simplification step, by Corollary 5.3.23. Lemma 5.3.26 shows that there exists a prebraid $\mathcal{K}_3$ that can be reached from $\mathcal{K}_1$ and $\mathcal{K}_2$ by simplification. Since $\mathcal{K}_1$ and $\mathcal{K}_2$ are irreducible, these simplification steps are not strict. Hence $\mathcal{K}_1$, $\mathcal{K}_2$ and $\mathcal{K}_3$ are variants, by Lemma 5.3.12 and Lemma 5.3.24. ■

Before we treat simplification of braids, let us mention three properties of irreducible prebraids.

Lemma 5.3.28 *(a) If the prebraid $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ is irreducible, then π_A and π_B are injective.*
(b) If $\mathcal{K}'$ is a subbraid of the irreducible prebraid $\mathcal{K}$, then $\mathcal{K}'$ is irreducible.
(c) If $\mathcal{K}_1$ and $\mathcal{K}_2$ are subbraids of the irreducible prebraid $\mathcal{K}$, and if $\mathcal{K}_1$ and $\mathcal{K}_2$ are variants, then $\mathcal{K}_1 = \mathcal{K}_2$.

Proof. (a) Assume that π_A, say, is not injective. Then there exist elements $\langle o_1, b_1 \rangle$ and $\langle o_2, b_1 \rangle$ in π_A where o_1 and o_2 are distinct. Let $m \in \mathcal{M}$ be an admissible endomorphism that maps o_1 to o_2 and leaves all other atoms fixed, let n be the identity on B. Now (m,n) is a strict simplifier for $\mathcal{K}$, thus we get a contradiction.

(b) Assume, to get a contradiction, that (m,n) is a strict simplifier for $\mathcal{K}'$. Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$, let $\mathcal{K}' = \langle a', C', D', \pi'_A, \pi'_B \rangle$. Let $X_A = \mathcal{O}_A(C) \setminus \mathcal{O}_A(C')$, let $Y_B = \mathcal{O}_B(D) \setminus \mathcal{O}_B(D')$. By Lemma 5.3.1 we may assume that m (n) leaves the elements of X_A (Y_B) fixed. If $\{m(o) \mid o \in \mathcal{O}_A(C')\} \cap X_A = \emptyset = \{n(o) \mid o \in \mathcal{O}_B(D')\} \cap Y_B$, then m (n) only identifies open atoms of $\mathcal{O}(\mathcal{K}')$ and it is easy to see that (m,n) is a strict simplifier for $\mathcal{K}$, which yields a contradiction. In the other case, let m' be an admissible automorphism such that $\{m'(m(o)) \mid o \in \mathcal{O}_A(C')\} \cap X_A = \emptyset$, let n' be an admissible automorphism such that $\{n'(n(o)) \mid o \in \mathcal{O}_B(D')\} \cap Y_B = \emptyset$. Let m^* denote the endomorphism that coincides with $m' \circ m$ on $\mathcal{O}_A(C')$ and leaves all other open atoms fixed. Let n^* denote the endomorphism that coincides with $n' \circ n$ on $\mathcal{O}_B(D')$ and leaves all other open atoms fixed. Then (m^*, n^*) is a strict simplifier for $\mathcal{K}$, which yields a contradiction.

(c) Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$, let $\mathcal{K}_i = \langle a_i, C_i, D_i, \pi^i_A, \pi^i_B \rangle$ $(i = 1, 2)$. Assume that $\mathcal{K}_1$ and $\mathcal{K}_2$ are variants, but $\mathcal{K}_1 \neq \mathcal{K}_2$. There exists a pair of admissible automorphisms (m,n) such that $\mathcal{K}_2 = \mathcal{K}_1^{(m,n)}$. Without loss of generality we have $(\diamond)$: there exists an $o^* \in \mathcal{O}_A(C_1)$ such that $o^* \neq m(o^*) \in \mathcal{O}_A(C_2)$.

Consider an element $o_0^A \in \mathcal{O}_A(C_1)$. If all elements of the "orbit"

$$o_0^A,\ o_1^A := m(o_0^A),\ o_2^A := m(o_1^A),\ o_3^A := m(o_2^A)\ldots$$

are in $\mathcal{O}_A(C_1)$, then this sequence contains only a finite number of distinct elements, say, $o_0^A, \ldots, o_k^A$. In the other case, let k be the first index in the sequence $0, 1, \ldots$ such that $o_k^A \notin \mathcal{O}_A(C_1)$. This implies that $o_k^A \in \mathcal{O}_A(C_2)$. The set $\{o_0^A, \ldots, o_k^A\}$ is called the m-trace $tr_m(o_0^A)$ of o_0^A. Let $\sim_m$ be the smallest equivalence relation on $\mathcal{O}_A(C_1) \cup \mathcal{O}_A(C_2)$ such that $o \sim_m o'$ whenever o and o' both belong to the m-trace $tr(o^A)$ of the same element $o^A \in \mathcal{O}_A(C_1)$. Since m is an injective function, the equivalence classes of $\sim_m$ are just the maximal m-traces. For each equivalence class $[o^A]_{\sim_m}$, choose a representant $rep([o^A]_{\sim_m})$. Let $m_\infty \in \mathcal{M}$ be the admissible endomorphism that maps each $o^A \in \mathcal{O}_A(C_1) \cup \mathcal{O}_A(C_2)$ to the representant $rep([o^A]_{\sim_m})$ and leaves other atoms fixed. Since $o^A \in \mathcal{O}_A(C_1)$ implies that $m(o^A) \in \mathcal{O}_A(C_2)$, and since both atoms have the same representant, we know that $m_\infty(m(o^A)) = m_\infty(o^A)$ for all $o^A \in \mathcal{O}_A(C_1)$. This implies, by Lemma 5.3.1, that $m_\infty(m(a)) = m_\infty(a)$ for all $a \in C_1$.

Symmetrically, we may define the n-traces $tr_n(o^B)$ of elements $o^B \in \mathcal{O}_B(D_1)$, just by replacing $\mathcal{O}_A(C_i)$ by $\mathcal{O}_B(D_i)$ $(i = 1, 2)$ and m by n. We obtain the equivalence relation $\sim_n$ by "identifying" all elements that belong to the same n-trace $tr_n(o^B)$, for some $o^B \in \mathcal{O}_B(D_1)$. For each equivalence class $[o^B]_{\sim_n}$, choose a representant $rep([o^B]_{\sim_n})$. Let $n_\infty \in \mathcal{N}$ be the admissible endomorphism that maps each $o^B \in \mathcal{O}_B(D_1) \cup \mathcal{O}_B(D_2)$ to the representant $rep([o^B]_{\sim_n})$ and leaves other atoms fixed. We have $n_\infty(n(o^B)) = n_\infty(o^B)$ for all $o^B \in \mathcal{O}_B(D_1)$, and $n_\infty(n(b)) = n_\infty(b)$ for all $b \in D_1$.

We want to show that (m_∞, n_∞) is a simplifier for $\mathcal{K}$. Suppose that $m_\infty(o) = m_\infty(o')$ for open atoms $o \neq o' \in \mathcal{O}_A(\mathcal{K})$. We may assume that there exists a sequence $o = o_0^A, o_1^A, \ldots, o_r^A = o'$ of elements of $\mathcal{O}_A(C_1) \cup \mathcal{O}_A(C_2)$, where at least the elements $o_0^A, \ldots, o_{r-1}^A$ are in $\mathcal{O}_A(C_1)$, such that $o_i^A = m^i(o_0^A)$, for $0 \leq i \leq r$. Let $b_i := \pi_A(o_i^A)$ $(0 \leq i \leq r)$. Note that at least the elements $b_0, \ldots, b_{r-1}$ are in D_1 since $\mathcal{K}_1$ is a subbraid of $\mathcal{K}$ and π_A and π_A^1 coincide on $\mathcal{O}_A(C_1)$. Since $\langle o_0^A, b_0 \rangle \in \pi_A^1$ we know, by choice of (m, n), that $\langle m(o_0^A), n(b_0) \rangle \in \pi_A^2 \subseteq \pi_A$, which means that $b_1 = n(b_0)$. Similarly we see that $b_i = n^i(b_0)$ for $i = 0, \ldots, r$. But then we have

$$n_\infty(b_0) = n_\infty(n(b_0)) = n_\infty(b_1) = \ldots = n_\infty(b_{k-1}) = n_\infty(n(b_{r-1})) = n_\infty(b_r)$$

Thus n_∞ identifies the π_A-images of $o = o_0^A$ and $o' = o_r^A$. Symmetrically, if $n_\infty(o) = n_\infty(o')$ for atoms $o, o' \in \mathcal{O}_B(\mathcal{K})$, then m_∞ identifies the π_B-images of o and o'. Therefore (m_∞, n_∞) is in fact a simplifier for $\mathcal{K}$. But (m_∞, n_∞) is strict, by $(\diamond)$. This is a contradiction. Thus $\mathcal{K}_1 = \mathcal{K}_2$. ∎

We shall now turn to simplification of braids. First we shall show that the result of two consecutive simplification steps may be obtained by a single simplification, similarly as for prebraids. We have to adapt the notion of a pending atom to the new situation.

Definition 5.3.29 Let (m,n) be a simplifier for the braid $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$. Let $\mathcal{K}^{\langle m,n\rangle} = \langle a', C', D', \pi'_A, \pi'_B\rangle$. Then the set

$$(\{m(o) \mid o \in \mathcal{O}_A(C)\} \setminus \mathcal{O}_A(C')) \ \cup \ (\{n(o) \mid o \in \mathcal{O}_B(D)\} \setminus \mathcal{O}_B(D'))$$

is called the set of *pending atoms* of the simplification step from $\mathcal{K}$ to the braid image $\mathcal{K}^{\langle m,n\rangle}$.

Note that this is really a new notion. The set of pending atoms of the simplification step from $\mathcal{K}$ to the *braid* image $\mathcal{K}^{\langle m,n\rangle}$ is a *superset* of the set of pending atoms of the simplification step from $\mathcal{K}$ to the image $\mathcal{K}^{(m,n)}$, but both sets are not necessarily identical.

Lemma 5.3.30 *Let (m_1, n_1) be a simplifier for the braid $\mathcal{K}_0$, let (m_2, n_2) be a simplifier for its braid image $\mathcal{K}_1 := \mathcal{K}_0^{\langle m_1,n_1\rangle}$. Assume that m_2 and n_2 do not identify any pending atom of the simplification step leading from $\mathcal{K}_0$ to the braid image $\mathcal{K}_1$ with another atom. Then $(m_2 \circ m_1, n_2 \circ n_1)$ is a simplifier for $\mathcal{K}_0$ and $\mathcal{K}_0^{\langle m_2\circ m_1,n_2\circ n_1\rangle} = \mathcal{K}_1^{\langle m_2,n_2\rangle}$.*

Proof. Exactly as in the corresponding proof of Lemma 5.3.22 it follows that $(m_2 \circ m_1, n_2 \circ n_1)$ is a simplifier for $\mathcal{K}_0$. Our assumptions guarantee that (m_2, n_2) is also a simplifier for $\mathcal{K}_0^{(m_1,n_1)}$ such that m_2 and n_2 do not identify any pending atom of the simplification step leading from $\mathcal{K}_0$ to the image $\mathcal{K}_0^{(m_1,n_1)}$ with another atom. Hence Lemma 5.3.22 implies that $(\mathcal{K}_0^{(m_1,n_1)})^{(m_2,n_2)} = \mathcal{K}_0^{(m_2\circ m_1,n_2\circ n_1)}$. Now $\mathcal{K}_1 = \mathcal{K}_0^{\langle m_1,n_1\rangle}$ is a subbraid of the prebraid $\mathcal{K}_0^{(m_1,n_1)}$ and both have the same root. Lemma 5.3.19 shows that $\mathcal{K}_1^{\langle m_2,n_2\rangle}$ is the unique subbraid of $(\mathcal{K}_0^{(m_1,n_1)})^{(m_2,n_2)} = \mathcal{K}_0^{(m_2\circ m_1,n_2\circ n_1)}$ given by its root, namely $\mathcal{K}_0^{\langle m_2\circ m_1,n_2\circ n_1\rangle}$. ■

Corollary 5.3.31 *Let (m_1, n_1) be a simplifier for the braid $\mathcal{K}_0$, let (m_2, n_2) be a simplifier for the braid image $\mathcal{K}_1 = \mathcal{K}_0^{\langle m_1,n_1\rangle}$. Then there exists a simplifier (m, n) for $\mathcal{K}_0$ such that $\mathcal{K}_0^{\langle m,n\rangle} = \mathcal{K}_1^{\langle m_2,n_2\rangle}$.*

Proof. It follows from Lemma 5.3.1 that there exists a simplifier (m'_2, n'_2) of $\mathcal{K}_1$ such that (m'_2, n'_2) does not identify any pending atom of the simplification step leading from $\mathcal{K}_0$ to the braid image $\mathcal{K}_1$ with another atom, and $\mathcal{K}_1^{\langle m_2,n_2\rangle} = \mathcal{K}_1^{\langle m'_2,n'_2\rangle}$. Let $(m,n) := (m'_2 \circ m_1, n'_2 \circ n_1)$. Then, by the previous lemma, $\mathcal{K}_0^{\langle m,n\rangle} = \mathcal{K}_1^{\langle m'_2,n'_2\rangle} = \mathcal{K}_1^{\langle m_2,n_2\rangle}$. ■

Theorem 5.3.32 *Let $\mathcal{K} = \mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_k$ be a sequence of braids such that each braid $\mathcal{K}_{i+1}$ is the braid image of $\mathcal{K}_i$ under a strict simplification, for $i = 0, \ldots, k-1$. Then $k \leq |\mathcal{O}(\mathcal{K})|$. If $\mathcal{K}'$ is an irreducible braid that is reached from $\mathcal{K}$ by a sequence of consecutive simplification steps (always taking braid images), then there exists a simplifier (m,n) for $\mathcal{K}$ such that $\mathcal{K}^{\langle m,n\rangle} = \mathcal{K}'$. If two irreducible braids $\mathcal{K}_1$ and $\mathcal{K}_2$ can be reached from $\mathcal{K}$ by sequences of consecutive simplification steps (always taking braid images), then $\mathcal{K}_1$ and $\mathcal{K}_2$ are variants.*

Proof. The first statement is trivial. The second statement follows from Corollary 5.3.31 by a simple induction. Assume that $\mathcal{K}_1$ and $\mathcal{K}_2$ are two irreducible braids that can be reached from $\mathcal{K}$ by sequences of consecutive simplification steps, always taking braid images. Then there exist simplifiers (m_1, n_1) and (m_2, n_2) of $\mathcal{K}$ such that $\mathcal{K}_1 = \mathcal{K}^{\langle m_1, n_1 \rangle}$ and $\mathcal{K}_2 = \mathcal{K}^{\langle m_2, n_2 \rangle}$. The prebraids $\mathcal{K}^{(m_1,n_1)}$ and $\mathcal{K}^{(m_2,n_2)}$ are not necessarily irreducible. But we may add further simplification steps (m_1', n_1') and (m_2', n_2') such that $(\mathcal{K}^{(m_1,n_1)})^{(m_1',n_1')}$ and $(\mathcal{K}^{(m_2,n_2)})^{(m_2',n_2')}$ are irreducible. By Lemma 5.3.16, (m_1', n_1') and (m_2', n_2') are – obviously non-strict – simplifiers for $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively. It follows that $\mathcal{K}_1$ and $\mathcal{K}_1^{\langle m_1', n_1' \rangle}$ are variants, and similarly for $\mathcal{K}_2$ and $\mathcal{K}_2^{\langle m_2', n_2' \rangle}$. By Theorem 5.3.27, the two prebraids $(\mathcal{K}^{(m_1,n_1)})^{(m_1',n_1')}$ and $(\mathcal{K}^{(m_2,n_2)})^{(m_2',n_2')}$ are variants. By Lemma 5.3.11, the two subbraids given by their roots – which are, by Lemma 5.3.19, $\mathcal{K}_1^{\langle m_1', n_1' \rangle}$ and $\mathcal{K}_2^{\langle m_2', n_2' \rangle}$ – are variants. Hence $\mathcal{K}_1$ and $\mathcal{K}_2$ are variants, by Lemma 5.3.12. ∎

On the basis of Theorem 5.3.32 we may introduce the following equivalence relation on the set of all braids.

Definition 5.3.33 Two braids are called *equivalent* if they can be simplified to the same irreducible braid image. If $\mathcal{K}$ is a braid, $[\mathcal{K}]$ denotes the set of all braids that are equivalent to $\mathcal{K}$.

Since two braids that are variants are obviously equivalent it is easy to see that we get in fact an equivalence relation. Let us also mention the following simple consequence of Theorem 5.3.32:

Lemma 5.3.34 *If two irreducible braids are equivalent, they are variants.*

5.3.4 Standard Normalisation

In order to define the underlying domain of the rational amalgam we shall now introduce a standard normal form for each braid. Let $\mathcal{O}_A^*$ be a subset of the set $\mathcal{O}_A$ of open atoms of A^Σ that has the same cardinality as the set of all equivalence classes of non-trivial[5] braids of type B. Similarly, let $\mathcal{O}_B^*$ be a subset of the set $\mathcal{O}_B$ of open atoms of B^Δ that has the same cardinality as the set of all equivalence classes of non-trivial braids of type A. Let $\mathfrak{A}_*^\Sigma := \mathrm{SH}_{\mathcal{M}}^{\mathfrak{A}}(Z \cup \mathcal{O}_A^*)$, and let $\mathfrak{B}_*^\Delta := \mathrm{SH}_{\mathcal{N}}^{\mathfrak{B}}(Z \cup \mathcal{O}_B^*)$. Lemma 3.2.13 shows

Lemma 5.3.35 *Every bijection between $Z \cup \mathcal{O}_A^*$ and $Z \cup \mathcal{O}_A$ extends to a Σ-isomorphism between $\mathfrak{A}_*^\Sigma$ and $\mathfrak{A}^\Sigma$. Similarly every bijection between $Z \cup \mathcal{O}_B^*$ and $Z \cup \mathcal{O}_B$ extends to a Δ-isomorphism between $\mathfrak{B}_*^\Delta$ and $\mathfrak{B}^\Delta$.*

We may now enumerate the elements of $\mathcal{O}_A^*$ and of $\mathcal{O}_B^*$ in the form

$$\begin{aligned}
\mathcal{O}_A^* &= \{o_{[\mathcal{K}]} \mid \mathcal{K} \text{ is a nontrivial braid of type } B\}, \\
\mathcal{O}_B^* &= \{o_{[\mathcal{K}]} \mid \mathcal{K} \text{ is a nontrivial braid of type } A\}.
\end{aligned}$$

[5] Compare Definition 5.3.3.

This means that $[\mathcal{K}] \mapsto o_{[\mathcal{K}]}$ establishes a bijection between the set of all equivalence classes of non-trivial braids of type A (B) and $\mathcal{O}_B^*$ ($\mathcal{O}_A^*$).

Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ be a prebraid. For each open atom $o \in \mathcal{O}_A(C)$ ($o \in \mathcal{O}_B(D)$) we say that *o points in $\mathcal{K}$ to $\mathcal{K}'$* iff $\mathcal{K}'$ is the unique subbraid of $\mathcal{K}$ with root $\pi_A(o)$ ($\pi_B(o)$)[6].

Definition 5.3.36 A prebraid $\mathcal{K}$ is in *standard normal form* if $\mathcal{O}_A(\mathcal{K}) \cup \mathcal{O}_B(\mathcal{K}) \subseteq \mathcal{O}_A^* \cup \mathcal{O}_B^*$ and if every open atom $o \in \mathcal{O}_A(\mathcal{K}) \cup \mathcal{O}_B(\mathcal{K})$ points in $\mathcal{K}$ to a subbraid $\mathcal{K}'$ such that $o = o_{[\mathcal{K}']}$.

With $A \odot B$ we denote the set of all braids over $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ in standard normal form. Note that trivial braids are always in standard normal form. Note also that the elements of a prebraid in standard normal form are in $\mathfrak{A}_*^\Sigma \cup \mathfrak{B}_*^\Delta$ (this follows from the remarks after Lemma 5.2.2).

Lemma 5.3.37 *Every prebraid in standard normal form is irreducible.*

Proof. Let $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ be a prebraid in standard normal form. Assume, to get a contradiction, that (m, n) is a strict simplifier for $\mathcal{K}$. Without loss of generality we may assume that $m(o_1) = m(o_2)$ for distinct open atoms $o_1, o_2 \in \mathcal{O}_A(C)$. Let $d_i := \pi_A(o_i)$, and let $\mathcal{K}_i$ denote the subbraid of $\mathcal{K}$ with root d_i, for $i = 1, 2$. Since (m, n) is a simplifier, $n(d_1) = n(d_2)$. By Lemma 5.3.16, (m, n) is a simplifier for $\mathcal{K}_1$ and $\mathcal{K}_2$. By Lemma 5.3.19, $\mathcal{K}_i^{\langle m,n \rangle}$ is the unique subbraid of $\mathcal{K}^{(m,n)}$ with root $n(d_i)$, for $i = 1, 2$. Since $n(d_1) = n(d_2)$, also $\mathcal{K}_1^{\langle m,n \rangle} = \mathcal{K}_2^{\langle m,n \rangle}$ which implies that $\mathcal{K}_1$ and $\mathcal{K}_2$ are equivalent. Since $\mathcal{K}$ is in standard normal form it follows that $o_1 = o_{[\mathcal{K}_1]} = o_{[\mathcal{K}_2]} = o_2$, which contradicts our assumption. ∎

Proposition 5.3.38 *Let $\mathcal{K}$ be a prebraid. Let (m, n) denote the admissible pair of endomorphisms that maps each $o \in \mathcal{O}_A(\mathcal{K}) \cup \mathcal{O}_B(\mathcal{K})$ to $o_{[\mathcal{K}']}$ where $\mathcal{K}'$ is the unique subbraid of $\mathcal{K}$ such that o points to $\mathcal{K}'$. Then (m, n) is a simplifier for $\mathcal{K}$ and $\mathcal{K}^{(m,n)}$ is in standard normal form.*

Proof. Let (m_1, n_1) be a simplifier for $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B \rangle$ such that $\mathcal{K}_1 := \mathcal{K}^{(m_1,n_1)}$ is irreducible. If m_1 identifies the open atoms $o, o' \in \mathcal{O}_A(C)$, then n_1 identifies $d := \pi_A(o)$ and $d' := \pi_A(o')$. It follows that o and o' point in $\mathcal{K}$ to subbraids that receive the same braid image under the simplification (m_1, n_1). Hence these subbraids are equivalent, which implies that $m(o) = m(o')$. It follows that the mapping $m_2 : m_1(o) \mapsto m(o)$ ($o \in \mathcal{O}_A(C)$) is well-defined. Symmetrically it follows that the mapping $n_2 : n_1(o) \mapsto n(o)$ ($o \in \mathcal{O}_B(D)$) is well-defined. Both mappings can be extended to admissible endomorphisms for which we shall use the same symbols.

[6] Compare Lemma 5.3.9.

Obviously $m_2 \circ m_1$ (resp. $n_2 \circ n_1$) and m (resp. n) coincide on $\mathcal{O}_A(C)$ (resp. $\mathcal{O}_B(D)$). Hence $m_2 \circ m_1$ (resp. $n_2 \circ n_1$) and m (resp. n) coincide on C (resp. D), by Lemma 5.3.1.

We shall now show that (m, n) is a simplifier for $\mathcal{K}$. Assume that $m(o) = m(o')$, for $o, o' \in \mathcal{O}_A(C)$. This means, by the definition of m, that o and o' point in $\mathcal{K}$ to equivalent subbraids $\mathcal{K}'$ and $\mathcal{K}''$, with roots $d := \pi_A(o)$ and $d' := \pi_A(o')$. We want to show that o and o' are already identified by m_1. Let $\mathcal{K}_1'$ and $\mathcal{K}_1''$ denote the braid images of $\mathcal{K}'$ and $\mathcal{K}''$ under the simplification (m_1, n_1) respectively. Then $\mathcal{K}'$ and $\mathcal{K}_1'$ are equivalent, and similarly for $\mathcal{K}''$ and $\mathcal{K}_1''$. Since $\mathcal{K}'$ and $\mathcal{K}''$ are equivalent, this implies that $\mathcal{K}_1'$ and $\mathcal{K}_1''$ are equivalent. But $\mathcal{K}_1'$ and $\mathcal{K}_1''$ are subbraids of the irreducible prebraid $\mathcal{K}_1$, by Lemma 5.3.19. Part (b) of Lemma 5.3.28 shows that $\mathcal{K}_1'$ and $\mathcal{K}_1''$ are irreducible. Since they are equivalent, both are variants, by Lemma 5.3.34. Part (c) of Lemma 5.3.28 shows that $\mathcal{K}_1' = \mathcal{K}_1''$. The root of $\mathcal{K}_1'$ is $n_1(d)$, and the root of $\mathcal{K}_1''$ is $n_1(d')$. Hence $n_1(d) = n_1(d')$ and $n(d) = n_2(n_1(d)) = n_2(n_1(d')) = n(d')$.

Symmetrically it follows that $n(o) = n(o')$ always implies that $m(\pi_B(o)) = m(\pi_B(o'))$, for all $o, o' \in \mathcal{O}_B(D)$. This shows that (m, n) is in fact a simplifier for $\mathcal{K}$. Obviously $\mathcal{K}^{(m,n)}$ is in standard normal form. ∎

Definition 5.3.39 The process where we apply to a given (pre)braid $\mathcal{K}$ the simplifier (m, n) that maps each open atom $o \in \mathcal{O}(\mathcal{K})$, pointing in $\mathcal{K}$ to the subbraid $\mathcal{K}'$, to the open atom $o_{[\mathcal{K}']} \in \mathcal{O}_A^* \cup \mathcal{O}_B^*$ will be called *standard simplification* of $\mathcal{K}$. The prebraid $\mathcal{K}^{(m,n)}$ (braid $\mathcal{K}^{\langle m,n\rangle}$) will be called *the standard (braid) normal form* of $\mathcal{K}$.

Obviously all subbraids of a prebraid in standard normal form are again in standard normal form.

Lemma 5.3.40 *For each braid $\mathcal{K}$ there exists exactly one braid $\mathcal{K}'$ in standard normal form such that $\mathcal{K}$ and $\mathcal{K}'$ are equivalent.*

Proof. We have seen that standard normalisation yields a braid in standard normal form that is equivalent to $\mathcal{K}$. If $\mathcal{K}'$ and $\mathcal{K}''$ are braids in standard normal form that are equivalent to $\mathcal{K}$, then $\mathcal{K}'$ and $\mathcal{K}''$ are irreducible (Lemma 5.3.37) and variants, by Lemma 5.3.34. It follows that there exists an admissible pair of automorphisms (m, n) such that $\mathcal{K}'' = \mathcal{K}'^{\langle m,n\rangle}$. Let $o \in \mathcal{O}_A(\mathcal{K}')$ point in $\mathcal{K}'$ to $\mathcal{K}_1$. Then $m(o)$ points in $\mathcal{K}''$ to $\mathcal{K}_1^{\langle m,n\rangle}$ and $\mathcal{K}_1$ and $\mathcal{K}_1^{\langle m,n\rangle}$ are equivalent. Since $\mathcal{K}'$ and $\mathcal{K}''$ are in standard normal form, $o = o_{[\mathcal{K}_1]} = o_{[\mathcal{K}_1^{\langle m,n\rangle}]} = m(o)$. Hence m coincides on the elements of $\mathcal{K}'$ of type A with identity. A symmetrical argument shows that n coincides on the elements of $\mathcal{K}'$ of type B with identity. ∎

Definition 5.3.41 Let $o \in \mathcal{O}_A^* \cup \mathcal{O}_B^*$. We say that o *represents* the unique braid $\mathcal{K}$ in standard normal form such that $o = o_{[\mathcal{K}]}$.

Lemma 5.3.42 *Given $e \in (A_* \cup B_*) \setminus (\mathcal{O}_A^* \cup \mathcal{O}_B^*)$ there exists a unique braid $\mathcal{K} \in A \odot B$ such that e is the root of $\mathcal{K}$.*

Proof. Let $e \in (A_* \cup B_*) \setminus (\mathcal{O}_A^* \cup \mathcal{O}_B^*)$. We may assume that $e \in A_* \setminus \mathcal{O}_A^*$. Let $\mathcal{O}_A(e) = \{o_1, \ldots, o_n\}$, and let o_i represent the braid in standard normal form $\mathcal{K}_i = \langle e_i, C_i, D_i, \pi_A^i, \pi_B^i \rangle$. Let $C := \bigcup_{i=1}^n C_i \cup \{e\}, D := \bigcup_{i=1}^n D_i, \pi_A := \bigcup_{i=1}^n \pi_A^i \cup \{\langle o_{[\mathcal{K}_i]}, e_i \rangle \mid i = 1, \ldots, n\}$, and $\pi_B := \bigcup_{i=1}^n \pi_B^i$. Then $\mathcal{K} = \langle e, C, D, \pi_A, \pi_B \rangle \in A \odot B$ has root e.

Conversely, let $\mathcal{K} = \langle e, C, D, \pi_A, \pi_B \rangle \in A \odot B$. Since each open atom o_i in $\mathcal{O}_A(e)$ represents a unique braid $\mathcal{K}_i$ to which it points in $\mathcal{K}$, the structure of $\mathcal{K}$ is completely determined by e. ∎

5.4 The Rational Amalgamated Product

In the first part of this section we introduce functions and relations on $A \odot B$ that interpret the symbols of the joint signature $\Sigma \cup \Delta$. With this step, the definition of the rational amalgamated product is complete. In the second subsection we add some evidence for the naturalness of rational amalgamation. We consider the case where the two components are non-collapsing quasi-free structures $(\mathfrak{A}^\Sigma, X, End_{\mathfrak{A}}^\Sigma)$ and $(\mathfrak{B}^\Delta, Y, End_{\mathfrak{B}}^\Delta)$ over disjoint signatures. This is the situation where we can build both the free amalgam and the rational amalgam with our actual methods.

Theorem 5.4.1 *The free amalgamated product is – modulo isomorphism – a substructure of the rational amalgamated product.*

This shows that there are interesting relationships between distinct amalgamation constructions.

We consider also a particular class of amalgamation components.

Theorem 5.4.2 *The rational amalgamated product of two algebras of rational trees over disjoint signatures is isomorphic to the algebra of rational trees over the combined signature.*

This shows that our general construction, complicated as it might appear, yields the expected result when we consider more concrete situations. The proof of the theorem is due to K. U. Schulz and can be found in [97, 98].

5.4.1 Functions and Relations

Given the underlying domain of the rational amalgam of $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ as constructed above, there is now a perfectly natural way to introduce functions and relations that interpret the symbols of the mixed signature $\Sigma \cup \Delta$. Basically, we shall use bijections to "copy" the Σ-structure (Δ-structure) of $\mathfrak{A}_*^\Sigma$ ($\mathfrak{B}_*^\Delta$) onto $A \odot B$. Consider the functions $root_A : A \odot B \to A_*$ and $root_B : A \odot B \to B_*$:

$$\begin{aligned} root_A(\mathcal{K}) &:= \begin{cases} \text{the root of } \mathcal{K} & \text{if } \mathcal{K} \text{ is trivial or has type } A \\ o_{[\mathcal{K}]} \in \mathcal{O}_A^* & \text{if } \mathcal{K} \text{ is non-trivial and has type } B. \end{cases} \\ root_B(\mathcal{K}) &:= \begin{cases} \text{the root of } \mathcal{K} & \text{if } \mathcal{K} \text{ is trivial or has type } B \\ o_{[\mathcal{K}]} \in \mathcal{O}_B^* & \text{if } \mathcal{K} \text{ is non-trivial and has type } A. \end{cases} \end{aligned}$$

As a direct consequence of Lemma 5.3.42 we obtain

Lemma 5.4.3 *The functions* $root_A$ *and* $root_B$ *are bijections.*

Here is now the definition of the rational amalgamated product.

Definition 5.4.4 The *rational amalgamated product* $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ *of* $\mathfrak{A}^\Sigma$ *and* $\mathfrak{B}^\Delta$ is the following $(\Sigma \cup \Delta)$-structure with carrier $A \odot B$:

1. Let $f \in \Sigma$ be an n-ary function symbol, let $\mathcal{K}_1, \ldots, \mathcal{K}_n \in A \odot B$. We define $f_{\mathfrak{A}\odot\mathfrak{B}}(\mathcal{K}_1, \ldots, \mathcal{K}_n) = root_A^{-1}(f_{\mathfrak{A}_*}(root_A(\mathcal{K}_1), \ldots, root_A(\mathcal{K}_n)))$.
2. Let $p \in \Sigma$ be an n-ary predicate symbol, let $\mathcal{K}_1, \ldots, \mathcal{K}_n \in A \odot B$. We define $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta \models p(\mathcal{K}_1, \ldots, \mathcal{K}_n)$ iff $\mathfrak{A}_*^\Sigma \models p(root_A(\mathcal{K}_1), \ldots, root_A(\mathcal{K}_n))$.

The interpretation of the function symbols $g \in \Delta$ and the predicate symbols $q \in \Delta$ in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ is defined symmetrically, using $root_B$.

Theorem 5.4.5 *As a* Σ*-structure,* $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta, \mathfrak{A}^\Sigma$ *and* $\mathfrak{A}_*^\Sigma$ *are isomorphic, and* $root_A : \mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta \to \mathfrak{A}_*^\Sigma$ *is a* Σ*-isomorphism. As a* Δ*-structure,* $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta, \mathfrak{B}^\Delta$*, and* $\mathfrak{B}_*^\Delta$ *are isomorphic, and* $root_B : \mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta \to \mathfrak{B}_*^\Delta$ *is a* Δ*-isomorphism.*

Proof. Recall that $\mathfrak{A}^\Sigma$ and $\mathfrak{A}_*^\Sigma$ are isomorphic, and similarly for $\mathfrak{B}^\Delta$ and $\mathfrak{B}_*^\Delta$. Lemma 5.4.3 and Definition 5.4.4 imply that $root_A : \mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta \to \mathfrak{A}_*^\Sigma$ is a Σ-isomorphism and $root_B : \mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta \to \mathfrak{B}_*^\Delta$ is a Δ-isomorphism. ∎

Theorem 5.4.5 makes clear that rational amalgamation is *not* a construction that can be used, say, to construct a rational tree algebra for a given signature Σ out of the finite tree algebra for Σ. Even if $\mathfrak{B}^\Delta$ consists of atoms only, the rational amalgam $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$, considered as a Σ-structure, is isomorphic to $\mathfrak{A}^\Sigma$.

5.4.2 Free Amalgamation and Rational Amalgamation

In this subsection we shall prove Theorem 5.4.1. We define the notion of an *acyclic* braid and show that the set of all acyclic braids in standard normal form is a substructure of the rational amalgamated product. It is then possible to prove that the free amalgamated product of the two component structures is isomorphic to this substructure.[7]

Definition 5.4.6 A prebraid $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$ is called *acyclic* if there is no sequence $e_1, e_2 \ldots, e_n$ of elements in $C \cup D$, of length $n \geq 2$, such that $e_1 = e_n$ and every element e_i is directly linked[8] via $\pi = \pi_A \cup \pi_B$ to e_{i+1}, for $i = 1, \ldots, n-1$. If $\mathcal{K}$ is acyclic, the *depth* of $\mathcal{K}$ is the largest number n such that there is a sequence $e_1, \ldots, e_n$ of elements of $\mathcal{K}$ where each element e_i is directly linked to e_{i+1} via π, for $i = 1, \ldots, n-1$. We write AC for the set of acyclic braids.

Lemma 5.4.7 *Let (m, n) be a simplifier for the acyclic braid $\mathcal{K}$. Then the braid image $\mathcal{K}^{\langle m,n\rangle}$ is an acyclic braid.*

Proof. We may assume that $\mathcal{K} = \langle a, C, D, \pi_A, \pi_B\rangle$ is of type A. Let $\mathcal{K}^{(m,n)} = \langle m(a), C', D', \pi'_A, \pi'_B\rangle$. We show that the prebraid $\mathcal{K}^{(m,n)}$ is acyclic. Assume, to get a contradiction, that there is a sequence $e_1 \ldots, e_n$ of elements in $C' \cup D'$, of length $n \geq 2$, such that $e_1 = e_n$ and every element e_i is directly linked to e_{i+1} via $\pi' := \pi'_A \cup \pi'_B$, for $i = 1, \ldots, n-1$. An element a of $\mathcal{K}$ is called interesting if its image $m(a)$ resp. $n(a)$ occurs in the sequence $e_1 \ldots, e_n$. An element b' of $\mathcal{K}$ is called a daughter of an element b of $\mathcal{K}$ if b' is directly linked to b via $\pi := \pi_A \cup \pi_B$.

Since $\mathcal{K}$ is acyclic, there has to be an interesting element $a \in C \cup D$ such that no daughter of a is interesting. Without loss of generality we assume that $a \in C$. Hence $m(a)$ occurs in $e_1 \ldots, e_n$, say, as element e_i (we may assume that $i > 1$). Since e_{i-1} is directly linked to $m(a) = e_i$ in $\mathcal{K}^{(m,n)}$, there exists a link $\langle o, e_{i-1}\rangle \in \pi'_A$ where $o \in \mathcal{O}_A(m(a))$. Let $\mathcal{O}_A(a) = \{o_1, \ldots, o_k\}$ and $b_i := \pi_A(o_i)$, for $i = 1, \ldots, k$. Thus $\{b_1, \ldots, b_k\}$ is the set of daughters of a in $\mathcal{K}$. From Lemma 5.2.4 it follows that o has the form $o = m(o_i)$, for some $1 \leq i \leq k$. Hence, since π'_A is a function, it follows from Definition 5.3.17 that $\langle o, e_{i-1}\rangle = \langle m(o_i), n(b_i)\rangle$ and $n(b_i) = e_{i-1}$. But this implies that the daughter b_i of a is interesting, which contradicts our choice of a. ■

Proposition 5.4.8 *The set AC of all acyclic braids of $A \odot B$ forms a substructure of $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$.*

Proof. Let $f \in \Sigma$ be an n-ary function symbol, let $\mathcal{K}_1, \ldots, \mathcal{K}_n$ be acyclic elements of $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. We have to show that the braid

$$f_{\mathfrak{A}\odot\mathfrak{B}}(\mathcal{K}_1, \ldots, \mathcal{K}_n) = root_A^{-1}(f_{\mathfrak{A}_*}(root_A(\mathcal{K}_1), \ldots, root_A(\mathcal{K}_n)))$$

[7] With the actual methods, the free amalgamated product can only be built for quasi-free structures $(\mathfrak{A}^\Sigma, X, \mathcal{M})$ and $(\mathfrak{B}^\Delta, Y, \mathcal{N})$ over disjoint signatures, where $\mathcal{M} = End^\Sigma_{\mathfrak{A}}$ and $\mathcal{N} = End^\Delta_{\mathfrak{B}}$. Hence we have to assume that the two components are non-collapsing and $\mathcal{M} = End^\Sigma_{\mathfrak{A}}$ and $\mathcal{N} = End^\Delta_{\mathfrak{B}}$.

[8] Compare Definition 5.3.2.

is acyclic. The elements of $\mathcal{O}_A(\{root_A(\mathcal{K}_1),\ldots,root_A(\mathcal{K}_n)\}$ represent – in the sense of Definition 5.3.41 – acyclic subbraids. By Lemma 5.2.3,

$$\mathcal{O}_A(f_{\mathfrak{A}_*}(root_A(\mathcal{K}_1),\ldots,root_A(\mathcal{K}_n)) \subseteq \mathcal{O}_A(\{root_A(\mathcal{K}_1),\ldots,root_A(\mathcal{K}_n)\}$$

and the open atoms in $a^* := f_{\mathfrak{A}_*}(root_A(\mathcal{K}_1),\ldots,root_A(\mathcal{K}_n))$ represent acyclic subbraids. If $a^* \in \mathfrak{A}_* \setminus \mathcal{O}_A^*$, then $\mathcal{K} := root_A^{-1}(a^*)$ is the unique braid in standard normal form with root a^*. Since all open atoms of the root of $\mathcal{K}$ represent acyclic subbraids, $\mathcal{K}$ itself is acyclic. In the other case, if $a^* = o \in \mathcal{O}_A(\{root_A(\mathcal{K}_1),\ldots,root_A(\mathcal{K}_n)\}$ is an atom, then it represents an acyclic braid in standard normal form $\mathcal{K}$ of type B. But $\mathcal{K} := root_A^{-1}(a^*)$. We have seen that the set of all acyclic braids represents a Σ-substructure of $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. Symmetrically it follows that this set represents a Δ-substructure of $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. ∎

We use the symbol $\mathcal{AC}$ for the substructure of acyclic braids.

We consider the amalgamation base $(Z, (\mathfrak{A}^\Sigma, Z), (\mathfrak{B}^\Delta, Z))$. Braids over $(\mathfrak{A}^\Sigma, Z)$ and $(\mathfrak{B}^\Delta, Z)$ are constructed by proceeding to non-collapsing superstructures $(\mathfrak{A}^\Sigma_\infty, Z \cup \mathcal{O}_A)$ and $(\mathfrak{B}^\Delta_\infty, Z \cup \mathcal{O}_B)$ where $\mathcal{O}_A$ ($\mathcal{O}_B$) are the open atoms of $\mathfrak{A}^\Sigma_\infty$ ($\mathfrak{B}^\Delta_\infty$). We have to show that there are homomorphisms $e_A : (\mathfrak{A}^\Sigma, Z) \to \mathcal{AC}$ and $e_B : (\mathfrak{B}^\Delta, Z) \to \mathcal{AC}$ such that $(\mathcal{AC}, e_A, e_B)$ closes the amalgamation base, i.e., $\mathcal{AC}$ is an amalgamation. Secondly, we have to show that it is the "smallest" amalgamation in the following sense. Let $(\mathfrak{D}, d_A, d_B)$ be another amalgamation where $(\mathfrak{A}^\Sigma, Z)$ is quasi-free for $\mathfrak{D}^\Sigma$ and $(\mathfrak{B}^\Delta, Z)$ is quasi-free for $\mathfrak{D}^\Delta$. Then there exists a unique homomorphism $h : \mathcal{AC} \to \mathfrak{D}$ such that $h \circ e_A = d_A$ and $h \circ e_B = d_B$.

Definition 5.4.9 Define $\mathcal{U}_A := \{x \in \mathcal{O}_A \mid \exists a \in A_\infty : x \in \mathrm{Stab}^{\mathfrak{A}_\infty}(a), root^{-1}_{A_\infty}(a) \in AC\}$ the set of open atoms occurring in acyclic braids and analogously $\mathcal{U}_B := \{x \in \mathcal{O}_B \mid \exists b \in B_\infty : x \in \mathrm{Stab}^{\mathfrak{B}_\infty}(b), root^{-1}_{B_\infty}(b) \in AC\}$. Define $A_\diamond := \mathrm{SH}^\Sigma_{A_\infty}(Z \cup \mathcal{U}_A)$ and $\mathfrak{A}^\Sigma_\diamond$ as its quasi-free structure; define $B_\diamond := \mathrm{SH}^\Delta_{B_\infty}(Z \cup \mathcal{U}_B)$ and $\mathfrak{B}^\Delta_\diamond$ as its quasi-free structure.

Lemma 5.4.10 *There exists a qf-isomorphism* $sm_A : (\mathfrak{A}^\Sigma_\infty, Z \cup \mathcal{O}_A) \to (\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ *and a qf-isomorphism* $sm_B : (\mathfrak{B}^\Delta_\infty, Z \cup \mathcal{O}_B) \to (\mathfrak{B}^\Delta_\diamond, Z \cup \mathcal{U}_B)$.

Proof. Since there exists a bijection between $Z \cup \mathcal{O}_A$ and $Z \cup \mathcal{U}_A$ (resp. between $Z \cup \mathcal{O}_B$ and $Z \cup \mathcal{U}_B$), this is an application of Lemmata 5.2 and 5.3, p. 27 in [11] which state the contents of our lemma in general terms. ∎

The elements of $A_\diamond$ and $B_\diamond$ are now mapped onto acyclic braids by means of the functions $root^{-1}_{A_\infty}$ and $root^{-1}_{B_\infty}$.

Lemma 5.4.11 *The function* $root^{-1}_{A_\infty}$ *is an isomorphism between* $\mathfrak{A}^\Sigma_\diamond$ *and* $\mathcal{AC}^\Sigma$. *The function* $root^{-1}_{B_\infty}$ *is an isomorphism between* $\mathfrak{B}^\Delta_\diamond$ *and* $\mathcal{AC}^\Delta$.

Proof. The function $root_{A_\infty}^{-1}$ is a homomorphism by definition, and so is its inverse $root_{A_\infty}$. All we need to show is that $root_{A_\infty}^{-1}$ is surjective onto AC^Σ. So let $a \in AC$ be an acyclic braid. If $a = \langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$ for some $z \in Z$, then clearly $a = root_{A_\infty}^{-1}(z)$. If a is of type A, then there is an $a' \in A_\infty$ with $a = root_{A_\infty}^{-1}(a')$ and $\mathrm{Stab}^{\mathfrak{A}_\infty}(a') \subseteq \mathcal{U}_A \cup Z$. Thus $\mathrm{Stab}^{\mathfrak{A}_\infty}(a') \subseteq A_\diamond$ and $a' \in A_\diamond$. If a is of type B, then there is an $o \in \mathcal{O}_A$ such that $a = root_{A_\infty}^{-1}(o)$ and $o \in \mathcal{U}_A \subseteq A_\diamond$ by definition.

The argument for $root_{B_\infty}^{-1}$ is analogue. ∎

Note, that $root_{A_\infty}^{-1}$ is not a *qf*-isomorphism. The elements of $\mathcal{U}_A$ are atoms in $\mathfrak{A}_\diamond^\Sigma$. Their images under $root_{A_\infty}^{-1}$ are complex braids and not the atoms $\langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$ where $z \in Z$.

Lemma 5.4.12 *The structure* $(\mathfrak{A}^\Sigma, Z)$ *is quasi-free for* $\mathcal{AC}^\Sigma$, $(\mathfrak{B}^\Delta, Z)$ *is quasi-free for* $\mathcal{AC}^\Delta$.

Proof. The structures $(\mathfrak{A}^\Sigma, Z)$ and $(\mathfrak{A}_\diamond^\Sigma, Z \cup \mathcal{U}_A)$ are qf-isomorphic, and $\mathfrak{A}_\diamond^\Sigma$ and $\mathcal{AC}^\Sigma$ are isomorphic. ∎

Now, we can define the embedding homomorphisms $e_A : (\mathfrak{A}^\Sigma, Z) \to \mathcal{AC}$ and $e_B : (\mathfrak{B}^\Delta, Z) \to \mathcal{AC}$. Consider the map $z \mapsto \langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$ for each $z \in Z$. Since $(\mathfrak{A}^\Sigma, Z)$ is quasi-free for $\mathcal{AC}^\Sigma$, there exists a unique extension to a homomorphism. We take e_A to be this unique extension. And e_B is the unique extension of that map from B to AC.

Lemma 5.4.13 *The triple* $(\mathcal{AC}, e_A, e_B)$ *closes the given amalgamation base* $(Z, (\mathfrak{A}^\Sigma, Z), (\mathfrak{B}^\Delta, Z))$.

Proof. The embedding homomorphism from Z to $(\mathfrak{A}^\Sigma, Z)$ is id_Z. And by definition of e_A, for every $z \in Z \subseteq A$ holds $e_A(z) = \langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$. id_Z is also the embedding homomorphism from Z to $(\mathfrak{B}^\Delta, Z)$. And $e_B(z) = \langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$, too. So clearly $e_1(z) = e_2(z)$. ∎

Proposition 5.4.14 $\mathcal{AC}$ *is the free amalgamated product of the amalgamation base* $(Z, (\mathfrak{A}^\Sigma, Z), (\mathfrak{B}^\Delta, Z))$.

Proof. Let $(\mathfrak{D}, d_A, d_B)$ be another amalgamation where $(\mathfrak{A}^\Sigma, Z)$ is quasi-free for $\mathfrak{D}^\Sigma$ and $(\mathfrak{B}^\Delta, Z)$ is quasi-free for $\mathfrak{D}^\Delta$. The functions d_A and d_B are the embedding Σ- (resp. Δ-) homomorphism from $\mathfrak{A}^\Sigma$ $(\mathfrak{B}^\Delta)$ to $\mathfrak{D}^\Sigma$ $(\mathfrak{D}^\Delta)$ such that d_A and d_B coincide on Z. We have to show that there exists a unique $\Sigma \cup \Delta$-homomorphism $h : \mathcal{AC}^{\Sigma\cup\Delta} \to \mathfrak{D}^{\Sigma\cup\Delta}$ that satisfies

$$d_A = h \circ e_A \text{ and } d_B = h \circ e_B.$$

The situation is illustrated by the following figure.

$$\begin{tikzcd}
 & \mathfrak{A}^{\Sigma} \arrow[dr, "e_A"] \arrow[drr, "d_A"] & & \\
Z \arrow[ur, "\mathrm{id}"] \arrow[dr, "\mathrm{id}"'] & & \mathcal{AC}^{\Sigma\cup\Delta} \arrow[r, "h!"] & \mathfrak{D}^{\Sigma\cup\Delta} \\
 & \mathfrak{B}^{\Delta} \arrow[ur, "e_B"] \arrow[urr, "d_B"'] & &
\end{tikzcd}$$

Since $(\mathfrak{A}^\Sigma, Z)$ and $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ are qf-isomorphic, $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ is also quasi-free for $\mathfrak{D}^\Sigma$ by Lemma 3.2.26. And $(\mathfrak{B}^\Delta_\diamond, Z \cup \mathcal{U}_B)$ is quasi-free for $\mathfrak{D}^\Delta$.

Definition of the homomorphism

For $k \geq 0$, define AC^k to be that subset of AC that contains all braids of depth at most k.

We start with the following simple observation. A given mapping $h^k : AC^k \to D$ induces two partial mappings $h^k_A := \{\langle root_{A_\infty}(a), d\rangle \mid \langle a, d\rangle \in h^k\} : A_\diamond \hookrightarrow D$ and $h^k_B := \{\langle root_{B_\infty}(a), d\rangle \mid \langle a, d\rangle \in h^k\} : B_\diamond \hookrightarrow D$.

We define an ascending tower of mappings $h^0 \subseteq h^1 \subseteq h^2 \ldots$ where $h^k : AC^k \to D$ for $k = 0, 1, 2, \ldots$. Thus we have $h^0_A \subseteq h^1_A \subseteq h^2_A \ldots$ and $h^0_B \subseteq h^1_B \subseteq h^2_B \ldots$. At each step of the construction of this tower, we will show that

$$\begin{array}{l} h^k_A \text{ can be extended to a } \Sigma\text{-homomorphism } g^{(k)}_{A_\diamond - D} : \mathfrak{A}^\Sigma_\diamond \to \mathfrak{D}^\Sigma, \\ h^k_B \text{ can be extended to a } \Delta\text{-homomorphism } g^{(k)}_{B_\diamond - D} : \mathfrak{B}^\Delta_\diamond \to \mathfrak{D}^\Delta. \end{array} \tag{5.1}$$

The construction will proceed by an induction over the depth of a braid.

For the base case, consider first a braid $\langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle$ where $z \in Z$. Define $h^0(\langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle) := d_A(z) = d_B(z)$. Now consider a braid $\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle$ of depth 0 and type A. Then $\mathrm{Stab}^{\mathfrak{A}_\infty}(a) \subseteq Z$. The mapping $h^0_A : Z \to D$ has an extension to a Σ-homomorphism from $\mathfrak{A}^\Sigma_\diamond$ to $\mathfrak{D}^\Sigma$, because $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ is quasi-free for $\mathfrak{D}^\Sigma$. And all such extensions to a Σ-homomorphism that agree on Z coincide on $\mathrm{SH}_{A_\diamond}(Z)$ by Lemma 3.2.24. Let $d \in D$ be the unique value that all extensions deliver. Set $h^0(\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle) := d$. If the braid $\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle$ of depth 0 is of type B, we define $h^0(\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle) := e$, the unique value of the extension of $h^0_B : Z \to D$ to a Δ-homomorphism from $\mathfrak{B}^\Delta_\diamond$ to $\mathfrak{D}^\Delta$.

We have to show that Condition (5.1) holds for h^0_A. But this is simple. For a braid $\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle$ of type A we just defined $h^0_A(a)$ to be the unique value of the homomorphic extension of h^0_A restricted to Z, and for a braid $\langle b, \emptyset, \{b\}, \emptyset, \emptyset\rangle$ of type B we know $root_{A_\infty}$ of it to be an atom in $\mathcal{U}_A$. So, since $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ is quasi-free for $\mathfrak{D}^\Sigma$ the mapping h^0_A can be extended to a Σ-homomorphism $g^{(0)}_{A_\diamond - D} : \mathfrak{A}^\Sigma_\diamond \to \mathfrak{D}^\Sigma$. Analogously, h^0_B can be extended to a Δ-homomorphism $g^{(0)}_{B_\diamond - D} : \mathfrak{B}^\Delta_\diamond \to \mathfrak{D}^\Delta$.

For the induction step, we suppose that h^k is already defined. Let $a = \langle a', E, F, \pi_E, \pi_F\rangle$ be a braid of type A of depth $k+1$. Then $a' = root_{A_\infty}(a)$ and $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a') \subseteq Z \cup \mathcal{U}_A$. Every element $u \in \mathrm{Stab}^{\mathfrak{A}_\diamond}(a') \cap \mathcal{U}_A$ is the A-name of an acyclic B-braid of depth at most k. Thus $h_A^k(u)$ is already defined. By induction hypothesis, h_A^k is extendable to a Σ-homomorphism $g_{A_\diamond - D}^{(k)} : \mathfrak{A}_\diamond^\Sigma \to \mathfrak{D}^\Sigma$. Define $h^{k+1}(a) := g_{A_\diamond - D}^{(k)}(a')$. This choice is not arbitrary. Since h_A^k is defined on the stabiliser of a', all extensions of h_A^k to a Σ-homomorphism must yield the same value for a' by Lemma 3.2.24. For braids b of depth $\leq k$, we define $h^{k+1}(b) := h^k(b)$.

We must show now, that Condition (5.1) holds for h_A^{k+1}. Let $V_A^k := (Z \cup \mathcal{U}_A) \cap \mathrm{dom}(h_A^k)$. So the set V_A^k is the restriction of the domain of h_A^k to the atoms. Let $h_{V_A^k - D}$ be the restriction of h_A^k to V_A^k. Since $(\mathfrak{A}_\diamond^\Sigma, Z \cup \mathcal{U}_A)$ is quasi-free for $\mathfrak{D}^\Sigma$, the mapping $h_{V_A^k - D} \cup \{\langle z, h_A^{k+1}(z)\rangle \mid z \in \mathrm{dom}(h_A^{k+1}) \setminus \mathrm{dom}(h_A^k)\}$ can be extended to a Σ-homomorphism $g_{A_\diamond - D}^{(k+1)} : \mathfrak{A}_\diamond^\Sigma \to \mathfrak{D}^\Sigma$. We will show that $g_{A_\diamond - D}^{(k+1)}$ extends h_A^{k+1}. Let $a \in \mathrm{dom}(h_A^k)$. Then by construction of h_A^k we know $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a) \subseteq \mathrm{dom}(h_A^k)$, and thus $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a) \subseteq V_A^k$. Hence h_A^{k+1} and $h_{V_A^k - D}$ agree on $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a)$. By Lemma 3.2.24 every homomorphic extension yields the same value, namely, $h_A^k(a)$. Let $a \in \mathrm{dom}(h_A^{k+1}) \setminus \mathrm{dom}(h_A^k)$ be a non-atom. The last paragraph's argument shows that $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a) \subseteq \mathrm{dom}(h_A^k)$; thus analogously to the above case, h_A^{k+1} and $h_{V_A^k - D}$ agree on $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a)$. Now $h_A^{k+1}(a)$ is exactly defined to be what all homomorphic extensions yield. Finally, let $z \in \mathrm{dom}(h_A^{k+1}) \setminus \mathrm{dom}(h_A^k)$ be an atom. Then immediately by definition of $g_{A_\diamond - D}^{(k+1)}$, we see $g_{A_\diamond}(z) = h_A^{k+1}(z)$. And that completes the proof that Condition (5.1) still holds for h_A^{k+1}. An analogous argument can be made for h_B^{k+1}.

We use the above described tower to define $h := \bigcup_{k \geq 0} h^k$. Let $h_A := \bigcup_{k \geq 0} h_A^k : A_\diamond \to D$ and $h_B := \bigcup_{k \geq 0} h_B^k : B_\diamond \to D$ be the mappings induced by h. We claim that h_A is a Σ-homomorphism. So, let $f \in \Sigma$ be an n-ary function symbol, $p \in \Sigma$ an n-ary predicate symbol, and $b_1, \ldots, b_n \in A_\diamond$. Choose k to be the maximum of the depths of $root_{A_\infty}^{-1}(b_1), \ldots, root_{A_\infty}^{-1}(b_n)$ and $root_{A_\infty}^{-1}(f(b_1, \ldots, b_n))$. Since h_A extends h_A^k, and since the latter mapping can be extended to a Σ-homomorphism $g_{A_\diamond - D}^{(k)}$, it follows that

$$\begin{array}{rcll}
h_A(f(b_1, \ldots, b_n)) & = & h_A^k(f(b_1, \ldots, b_n)) & \\
& = & g_{A_\diamond - D}^{(k)}(f(b_1, \ldots, b_n)) & \\
& = & f(g_{A_\diamond - D}^{(k)}(b_1), \ldots, g_{A_\diamond - D}^{(k)}(b_n)) & \\
& = & f(h_A^k(b_1), \ldots, h_A^k(b_n)) & \\
& = & f(h_A(b_1), \ldots, h_A(b_n)), & \text{and} \\
p(b_1, \ldots, b_n) & \Longrightarrow & p(g_{A_\diamond - D}^{(k)}(b_1), \ldots, g_{A_\diamond - D}^{(k)}(b_n)) & \\
& \Longleftrightarrow & p(h_A^k(b_1), \ldots, h_A^k(b_n)) & \\
& \Longleftrightarrow & p(h_A(b_1), \ldots, h_A(b_n)). &
\end{array}$$

A similar argument shows that h_B is a Δ-homomorphism. It remains to be shown that h is a $\Sigma \cup \Delta$-homomorphism and the factorising homomorphism. First note that $h = h_A \circ root_{A_\infty} = h_B \circ root_{B_\infty}$. In fact, let a be a braid of depth l. Then $h(a) = h^l(a) = h_A^l(root_{A_\infty}(a)) = h_A(root_{A_\infty}(a))$ and $h(a) =$

$h^l(a) = h^l_B(root_{B_\infty}(a)) = h_B(root_{B_\infty}(a))$.
Let $f \in \Sigma \cup \Delta$ be an n-ary function symbol, and $a_1, \ldots, a_n \in AC$. Suppose $f \in \Sigma$. (The case $f \in \Delta$ is similar.) Then

$$\begin{aligned}
h(f(a_1,\ldots,a_n)) &= h_A(root_{A_\infty}(f(a_1,\ldots,a_n))) \\
&= h_A(root_{A_\infty}(root^{-1}_{A_\infty}(f(root_{A_\infty}(a_1),\ldots,root_{A_\infty}(a_n))))) \\
&= h_A(f(root_{A_\infty}(a_1),\ldots,root_{A_\infty}(a_n))) \\
&= f(h_A(root_{A_\infty}(a_1)),\ldots,h_A(root_{A_\infty}(a_n))) \\
&= f(h(a_1),\ldots,h(a_n)).
\end{aligned}$$

Let $p \in \Sigma \cup \Delta$ be an n-ary predicate symbol, and $a_1, \ldots, a_n \in AC$. Again we suppose p to be in Σ without restricting generality. Then

$$\begin{aligned}
p(a_1,\ldots,a_n) &\iff p(root_{A_\infty}(a_1),\ldots,root_{A_\infty}(a_n)) \\
&\implies p(h_A(root_{A_\infty}(a_1)),\ldots,h_A(root_{A_\infty}(a_n))) \\
&\iff p(h(a_1),\ldots,h(a_n)).
\end{aligned}$$

The composition $h \circ e_A$ is a Σ-homomorphism from $(\mathfrak{A}^\Sigma, Z)$ to $\mathfrak{D}^\Sigma$. And for all $z \in Z$ we have $h(e_A(z)) = h(\langle z\{z\}, \emptyset, \emptyset, \emptyset\rangle) = d_A(z)$ by definition of e_A and h. Since $(\mathfrak{A}^\Sigma, Z)$ is quasi-free for $\mathfrak{D}^\Sigma$, it follows that $h \circ e_A$ and d_A coincide on A. Similarly we can show that $h \circ e_B$ and d_B coincide on B. This shows that h is the factoring homomorphism (or, categorically speaking, mediating morphism).

Uniqueness of the homomorphism
Assume that $g : \mathcal{AC}^{\Sigma\cup\Delta} \to \mathfrak{D}^{\Sigma\cup\Delta}$ is another $\Sigma \cup \Delta$-homomorphism such that

$$g \circ e_A = d_A \text{ and } g \circ e_B = d_B \tag{5.2}$$

g induces a Σ-homomorphisms $g_A : (\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A) \to \mathfrak{D}^\Sigma$ and a Δ-homomorphism $g_B : (\mathfrak{B}^\Delta_\diamond, Z \cup \mathcal{U}_B) \to \mathfrak{D}^\Delta$. We will show by induction on the depth of a braid that $g = h$.

By definition of e_A, e_B and (5.2), $g(\langle z, \{z\}, \emptyset, \emptyset, \emptyset\rangle) = d_1(z) = d_2(z)$ for all $z \in Z$. Thus g and h coincide on Z and also $g_A =_Z h_A, g_B =_Z h_B$. Let $a \notin Z$ and $\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle$ be a type A braid of depth 0. Then $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a) \subseteq Z$. Therefore g_A and h_A coincide on a, because $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ is quasi-free for $\mathfrak{D}^\Sigma$. Hence g and h coincide on $\langle a, \{a\}, \emptyset, \emptyset, \emptyset\rangle$.

Now suppose that g and h coincide on all braids with depth at most k. Consider a braid $a = \langle a', E, F, \pi_E, \pi_F\rangle$ of type A of depth $k+1$. Then $a' = root_{A_\infty}(a)$ and $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a') \subseteq Z \cup \mathcal{U}_A$. Every element $u \in \mathrm{Stab}^{\mathfrak{A}_\diamond}(a') \cap \mathcal{U}_A$ is the A-name of an acyclic B-braid of depth at most k. By induction hypothesis, g_A and h_A coincide on $\mathrm{Stab}^{\mathfrak{A}_\diamond}(a')$. Thus they coincide on a', because $(\mathfrak{A}^\Sigma_\diamond, Z \cup \mathcal{U}_A)$ is quasi-free for $\mathfrak{D}^\Sigma$. And hence g and h coincide on a. ∎

5.5 Combination of Constraint Solvers

Our last aim is to show how constraint solvers for two component structures can be combined to a constraint solver for their rational amalgamated product.

Constraint solvers, as considered here, are essentially algorithms that decide solvability of quantifier-free positive formulae in a given solution domain. We (mostly) disregard disjunction since its integration is a triviality.

Definition 5.5.1 Let Θ be a signature. A Θ-*constraint* is a conjunction of atomic Θ-formulae.

In order to decide solvability of a "mixed" $(\Sigma \cup \Delta)$-constraint in a rational amalgamated product $\mathfrak{A}^{\Sigma} \odot \mathfrak{B}^{\Delta}$ we shall decompose it into two pure constraints over the signatures Σ and Δ respectively. These output constraints are equipped with additional restrictions of a particular type.

Definition 5.5.2 An *A/N (atom/non-atom) declaration* for a constraint Γ is a pair (U, W) such that $U \uplus W \subseteq \mathit{Var}(\Gamma)$ is a disjoint union. Both U and W may be empty. A solution ν_A of a constraint Γ in a quasi-free structure $(\mathfrak{A}^{\Sigma}, X, \mathcal{M})$ is called a *solution* of $\langle \Gamma, U, W \rangle$ if ν_A assigns distinct atoms to the variables in U, and arbitrary non-atomic elements of A to the variables in W.

In order to avoid some ballast in proofs we shall assume that at least one of the two components is a *non-trivial* quasi-free structure, which means that it has at least one non-atomic element. We may now formulate our main result concerning combination of constraint solvers in the case of rational amalgamation.

Theorem 5.5.3 *Let $\mathfrak{A}^{\Sigma}$ and $\mathfrak{B}^{\Delta}$ be two non-collapsing quasi-free structures over disjoint signatures, let $\mathfrak{A}^{\Sigma} \odot \mathfrak{B}^{\Delta}$ denote their rational amalgam. Assume that at least one of the two components is a non-trivial quasi-free structure. Then solvability of $(\Sigma \cup \Delta)$-constraints in $\mathfrak{A}^{\Sigma} \odot \mathfrak{B}^{\Delta}$ is decidable if solvability of (Σ- resp. Δ-) constraints with A/N declarations is decidable for $\mathfrak{A}^{\Sigma}$ and $\mathfrak{B}^{\Delta}$.*

There seems to be no general way to characterise solvability of Γ-constraints with A/N declarations in purely logical terms. But for a restricted class of component structures – a class which is of particular interest in the context of rational amalgamation – a logical characterisation of the problems that we have to solve in the two component structures can be given.

Definition 5.5.4 A non-collapsing quasi-free structure $(\mathfrak{A}^{\Sigma}, X, \mathcal{M})$ is called *rational* if for every atom $x \in X$ and every element $a \in A$ there exists an endomorphism $m \in \mathcal{M}$ that leaves all atoms $x' \neq x$ fixed such that $m(x) = m(a)$.[9]

The algebra of rational trees over a given signature is always a rational quasi-free structure. The same holds for feature structures, feature structures with arity, and domains with nested, rational lists (as described in 3.2.17). For rational quasi-free structures we obtain the following refinement and reformulation of Theorem 5.5.3.

[9]The existence of such an endomorphism is trivial if $x \notin \mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(a)$. In this case we may always take, e.g., the endomorphism $m = m_{x-a}$ of $\mathcal{M}$ that maps x to a and leaves all other atoms fixed. The situation of interest is the case where $x \in \mathrm{Stab}^{\mathfrak{A}}_{\mathcal{M}}(a)$ and $x \neq a$.

Theorem 5.5.5 *Let $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ be two non-trivial rational quasi-free structures over disjoint signatures, let $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ denote their rational amalgam. Then solvability of $(\Sigma \cup \Delta)$-constraints in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ is decidable if the positive universal-existential theory is decidable for both components $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$.*

Since existential quantification distributes over disjunction, the theorem may be slightly strengthened.

Theorem 5.5.6 *Let $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ be two non-trivial rational quasi-free structures over disjoint signatures, let $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ denote their rational amalgam. Then the positive existential theory of $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ is decidable if the positive universal-existential theory is decidable for both components $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$.*

It is interesting to contrast this formulation with the corresponding combination result for free amalgamation. The assumptions on the components are stronger. In oder to decide the positive existential theory of the free amalgamted product, the positive theories of both components must be decidable (Theorem 3.4.2). On the other hand, the quantifier fragment that can be decided in the free amalgamted product is larger. As Theorem 3.4.1 states, the *full* positive theory can be decided in the free amalgamated product, provided the positive theories are decidable in the components.

One application of Theorem 5.5.6 is the following

Corollary 5.5.7 *Rational amalgamated products $\mathfrak{A}_1^{\Sigma_1} \odot \cdots \odot \mathfrak{A}_k^{\Sigma_k}$ have decidable positive existential theory if the nontrivial components $\mathfrak{A}_i^{\Sigma_i}$ are rational tree algebras, or nested, rational lists, or feature structures*[10]*, or feature-structures with arity, for $i = 1, \ldots, k$, and if the signatures of the components are pairwise disjoint.*

Proof. For all these structures it has been shown that even the full positive theory is decidable, see [10]. ∎

In the rest of this section, we prove Theorem 5.5.3 and Theorem 5.5.5.

5.5.1 Proof of Theorem 5.5.3

To prove Theorem 5.5.3 we shall give an algorithm that reduces a mixed constraint Γ in the signature $(\Sigma \cup \Delta)$ non-deterministically to a pair of constraints with A/N declarations over the "pure" signatures Σ and Δ respectively. We shall assume that the input formula Γ has the form $\Gamma = \Gamma_0^\Sigma \wedge \Gamma_0^\Delta$ where Γ_0^Σ is a conjunction of atomic Σ-formulae, and Γ_0^Δ is a conjunction of atomic Δ-formulae. Moreover we assume that Γ does not contain any equation between variables. These assumptions do not really restrict the generality of the approach. The first two steps of the decomposition algorithm in Section 3.4.1 on page 38 show that one can transform an arbitrary $(\Sigma \cup \Delta)$-constraint φ into a constraint Γ of the form given above, preserving solvability in both directions.

[10] As in Examples 3.2.17 we refer to [2], for specificity.

Algorithm 5.5.8 The *input* is mixed a constraint $\Gamma = \Gamma_0^\Sigma \wedge \Gamma_0^\Delta$ of the form described above. Let $V_0 = Var(\Gamma_0^\Sigma) \cap Var(\Gamma_0^\Delta)$ denote the set of *shared* variables of Γ. The algorithm has two steps, both are nondeterministic.

Step 1: Variable identification. *Consider all possible partitions of the set of all shared variables, V_0. Each of these partitions yields one of the new constraints as follows. The variables in each class of the partition are "identified" with each other by choosing an element of the class as representative, and replacing in the input formula all occurrences of variables of the class by this representative.*

Step 2: Choose signature labels. *Let $\Gamma_1^\Sigma \wedge \Gamma_1^\Delta$ denote one of the formulae obtained by Step 1, let V_1 denote the set of representants of shared variables. The set V_1 is partitioned in two subsets U and W in some arbitrary way.*

Let $\sigma = \Gamma_1^\Sigma$, let $\delta = \Gamma_1^\Delta$. For each of the choices made in Step 1 and 2, the algorithm yields an *output pair* $(\langle \sigma, U, W\rangle, \langle \delta, W, U\rangle)$, each component representing a constraint with A/N declaration.

Correctness of Algorithm 5.5.8

We shall prove that Algorithm 5.5.8 is correct in the following sense.

Proposition 5.5.9 *The input formula Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ if and only if there exists an output pair $(\langle \sigma, U, W\rangle, \langle \delta, W, U\rangle)$ of Algorithm 5.5.8 such that $\langle \sigma, U, W\rangle$ has a solution in $\mathfrak{A}^\Sigma$ and $\langle \delta, W, U\rangle$ has a solution in $\mathfrak{B}^\Delta$.*

Note that Theorem 5.5.3 is an immediate consequence. In order to prove Proposition 5.5.9 we shall assume that the two components $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ are quasi-free structures of the form $(\mathfrak{A}^\Sigma, X, \mathcal{M})$ and $(\mathfrak{B}^\Delta, Y, \mathcal{N})$ respectively. First we show soundness.

Lemma 5.5.10 *If, for some particular output pair $(\langle \sigma, U, W\rangle, \langle \delta, W, U\rangle)$ of Algorithm 5.5.8, $\langle \sigma, U, W\rangle$ has a solution in $\mathfrak{A}^\Sigma$ and $\langle \delta, W, U\rangle$ has a solution in $\mathfrak{B}^\Delta$, then the input constraint Γ is solvable in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$.*

Proof. The output formulae σ and δ may be written in the form $\Gamma_1^\Sigma(\vec{u}, \vec{w}, \vec{v}_\Sigma)$ and $\Gamma_1^\Delta(\vec{u}, \vec{w}, \vec{v}_\Delta)$, where $\vec{u} = u_1, \ldots, u_m$ denotes the sequence of all elements of U, where $\vec{w} = w_1, \ldots, w_n$ denotes the sequence of all elements of W, and where $\vec{v}_\Sigma$ (resp. $\vec{v}_\Delta$) stands for the non-shared variables occurring in Γ_1^Σ and Γ_1^Δ respectively. The proof has now three steps. In the first step, the given solutions of the output constraints are used to construct similar solutions of a more specific form. In the second step, these latter solutions are used to define suitable braids. In the third step we apply standard normalisation to these braids. This will yield a solution of the input constraint.

1. By assumption, there exists a solution μ_A of Γ_1^Σ in $\mathfrak{A}^\Sigma$ such that the elements $\mu_A(u_1), \ldots, \mu_A(u_m)$ are distinct atoms of $\mathfrak{A}^\Sigma$, and the elements

$\mu_A(w_1), \ldots, \mu_A(w_n)$ are non-atomic elements of $\mathfrak{A}^\Sigma$. If some of the atoms $\mu_A(u_1), \ldots, \mu_A(u_m)$ are bottom atoms, then we apply an automorphism $m_1 \in \mathcal{M}$ such that the elements in $\{m_1(\mu_A(u_1)), \ldots, m_1(\mu_A(u_m))\}$ are distinct open atoms. In the other case, let $m_1 := \mathrm{id}$. If the stabilisers of the elements $m_1(\mu_A(w_1)), \ldots, m_1(\mu_A(w_n))$ contain open atoms $o_1, \ldots, o_k$ that do not belong to $\{m_1(\mu_A(u_1)), \ldots, m_1(\mu_A(u_m))\}$, then we apply an endomorphism m_2 that maps the atoms $o_1, \ldots, o_k$ to some bottom atom z and leaves the atoms $\{m_1(\mu_A(u_1)), \ldots, m_1(\mu_A(u_m))\}$ fixed. In the other case, let $m_2 := \mathrm{id}$. Since Γ_1^Σ is a positive formula, $\nu_A := \mu_A \circ m_1 \circ m_2$ is a solution of Γ_1^Σ, by Lemma 5.2.1. We have

(1) the elements $x_1 := \nu_A(u_1), \ldots, x_m := \nu_A(u_m)$ are distinct *open* atoms,

(2) the elements $a_1 := \nu_A(w_1), \ldots, a_n := \nu_A(w_n)$ are non-atomic,

(3) the open atoms occurring in the stabilisers of the elements $a_1, \ldots, a_n$ are in $\{x_1, \ldots, x_m\}$, and

(4) $\mathfrak{A}^\Sigma \models \exists \vec{v}_\Sigma\, \Gamma_1^\Sigma(\vec{u}/\vec{x}, \vec{w}/\vec{a})$.

(2) follows from the fact $\mathfrak{A}^\Sigma$ is non-collapsing, (3) follows from Lemma 5.2.4, and (4) follows from the fact that ν_A solves Γ_1^Σ. Symmetrically we can show that there exists a solution ν_B of Γ_1^Δ in $\mathfrak{B}^\Delta$ such that

(5) the elements $y_1 := \nu_B(w_1), \ldots, y_n := \nu_B(w_n)$ are distinct *open* atoms,

(6) the elements $b_1 := \nu_B(u_1), \ldots, b_m := \nu_B(u_m)$ are non-atomic,

(7) the open atoms occurring in the stabilisers of the elements $b_1, \ldots, b_m$ are in $\{y_1, \ldots, y_n\}$, and

(8) $\mathfrak{B}^\Delta \models \exists \vec{v}_\Delta\, \Gamma_1^\Delta(\vec{u}/\vec{b}, \vec{w}/\vec{y})$.

2. Let $\pi_A := \{\langle x_i, b_i\rangle \mid i = 1, \ldots, m\}$, let $\pi_B := \{\langle y_i, a_i\rangle \mid i = 1, \ldots, n\}$. Properties (1)–(3) and (5)–(7) show that for each $e \in \vec{a}$ ($e \in \vec{b}$), the tuple $\mathcal{K}_e := \langle e, \{a_1, \ldots, a_n\}, \{b_1, \ldots, b_m\}, \pi_A, \pi_B\rangle$ is a prebraid of type A (B).

3. Fix some $e \in \vec{a} \cup \vec{b}$. Let (m_3, n_3) be the standard normaliser for $\mathcal{K}_e$. By Lemma 5.2.1, (4), and (8),

$$\begin{aligned}
\mathfrak{A}^\Sigma &\models \exists \vec{v}_\Sigma\, \Gamma_1^\Sigma(\vec{u}/m_3(\vec{x}), \vec{w}/m_3(\vec{a})),\\
\mathfrak{B}^\Delta &\models \exists \vec{v}_\Delta\, \Gamma_1^\Delta(\vec{u}/n_3(\vec{b}), \vec{w}/n_3(\vec{y})).
\end{aligned}$$

It follows easily from Lemma 5.3.35 that

$$\begin{aligned}
\mathfrak{A}_*^\Sigma &\models \exists \vec{v}_\Sigma\, \Gamma_1^\Sigma(\vec{u}/m_3(\vec{x}), \vec{w}/m_3(\vec{a})),\\
\mathfrak{B}_*^\Delta &\models \exists \vec{v}_\Delta\, \Gamma_1^\Delta(\vec{u}/n_3(\vec{b}), \vec{w}/n_3(\vec{y})).
\end{aligned}$$

Now Theorem 5.4.5 shows that

$$\begin{aligned}
\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta &\models \exists \vec{v}_\Sigma\, \Gamma_1^\Sigma(\vec{u}/root_A^{-1}(m_3(\vec{x})), \vec{w}/root_A^{-1}(m_3(\vec{a}))),\\
\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta &\models \exists \vec{v}_\Delta\, \Gamma_1^\Delta(\vec{u}/root_B^{-1}(n_3(\vec{b})), \vec{w}/root_B^{-1}(n_3(\vec{y}))).
\end{aligned}$$

Consider an element x_i of $\vec{x}$. Assume that x_i points in $\mathcal{K}_e$ to the subbraid $\mathcal{K}'$ with root b_i. Then $m_3(x_i) = o_{[\mathcal{K}']}$. Let $\mathcal{K}_i$ be the subbraid of $\mathcal{K}_e^{(m_3,n_3)}$ with root $n_3(b_i)$. By Lemma 5.3.19, $\mathcal{K}'$ and $\mathcal{K}_i$ are equivalent. It follows that $m_3(x_i) = o_{[\mathcal{K}_i]}$. The braid $\mathcal{K}_i$ is non-trivial and of type B, and it is the unique braid in standard normal form with root $n_3(b_i)$ (Prop. 5.3.38, Lemma 5.3.42). Hence $root_A^{-1}(m_3(x_i)) = \mathcal{K}_i$. The element $n_3(b_i)$ is a non-atomic element of B. Hence $root_B^{-1}(n_3(\vec{b})) = \mathcal{K}_i$ is the unique braid in standard normal form with root $n_3(b_i)$. Thus we have seen that $root_A^{-1}(m_3(\vec{x})) = root_B^{-1}(n_3(\vec{b}))$. Similarly it follows that $root_B^{-1}(n_3(\vec{y})) = root_A^{-1}(m_3(\vec{a}))$. This shows that the formula $\Gamma_1^\Sigma \wedge \Gamma_1^\Delta$ obtained after Step 1 has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. Obviously this implies that the input constraint Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. ∎

Next, we show completeness of the Algorithm 5.5.8.

Proposition 5.5.11 *If the input constraint Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$, then there exists an output pair $(\langle\sigma, U, W\rangle, \langle\delta, W, U\rangle)$ of Algorithm 5.5.8 such that $\langle\sigma, U, W\rangle$ has a solution in $\mathfrak{A}^\Sigma$ and $\langle\delta, W, U\rangle$ has a solution in $\mathfrak{B}^\Delta$.*

Proof. Assume that Γ has a solution $\mu_{A\odot B}$ in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$.

In Step 1 of Algorithm 5.5.8 we identify two shared variables v and v' of V_0 if, and only if, $\mu_{A\odot B}(v) = \mu_{A\odot B}(v')$. With this choice, $\mu_{A\odot B}$ is a solution of the formula $\Gamma_1^\Sigma \wedge \Gamma_1^\Delta$ that is reached after Step 1, and $\mu_{A\odot B}$ assigns distinct values in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ to all variables of V_1.

By Theorem 5.4.5, $root_A \circ \mu_{A\odot B}$ (resp. $root_B \circ \mu_{A\odot B}$) is a solution of $\sigma = \Gamma_1^\Sigma$ in $\mathfrak{A}_*^\Sigma$ (resp. of $\delta = \Gamma_1^\Delta$ in $\mathfrak{B}_*^\Delta$) that does not identify two variables of V_1.

By assumption, one of the two component structures, $\mathfrak{A}^\Sigma$, say, is non-trivial. In Step 2, we choose as U the set of all variables u of V_1 such that $\mu_{A\odot B}(u)$ is a non-trivial braid of type B. Consequently, W contains all variables w of V_1 such that $\mu_{A\odot B}(w)$ is a trivial braid or a non-trivial braid of type A. The definition of $root_A$ implies that $root_A \circ \mu_{A\odot B}(u)$ is an open atom of $\mathfrak{A}_*^\Sigma$, for all $u \in U$, and $root_A \circ \mu_{A\odot B}(w)$ is a non-atomic element or a bottom atom of $\mathfrak{A}_*^\Sigma$, for all $w \in W$. Let $m_1 \in \mathcal{M}$ be an endomorphism that maps all the bottom atoms of the set $\{root_A \circ \mu_{A\odot B}(w) \mid w \in W\}$ to a non-atomic element of A and leaves all other atoms fixed. Since $\mathfrak{A}^\Sigma$ is non-collapsing, all elements of the set $\{m_1 \circ root_A \circ \mu_{A\odot B}(w) \mid w \in W\}$ are non-atomic. Since σ is a positive formula, Lemma 5.2.1 implies that $\nu_A := m_1 \circ root_A \circ \mu_{A\odot B}$ is a solution of $\langle\sigma, U, W\rangle$ in $\mathfrak{A}_*^\Sigma$.

On the other hand the definition of $root_B$ implies that $root_B \circ \mu_{A\odot B}(w)$ is an atom of $\mathfrak{B}_*^\Delta$, for all $w \in W$, and $root_B \circ \mu_{A\odot B}(u)$ is a non-atomic element of $\mathfrak{B}_*^\Delta$, for all $u \in U$. This shows that $\langle\delta, W, U\rangle$ has a solution in $\mathfrak{B}_*^\Delta$.

But then, by Lemma 5.3.35, $\langle\sigma, U, W\rangle$ has a solution in $\mathfrak{A}^\Sigma$ and $\langle\delta, W, U\rangle$ has a solution in $\mathfrak{B}^\Delta$. ∎

5.5.2 An Example

In order to illustrate Algorithm 5.5.8, we discuss a little example problem in the combination of rational trees and non-wellfounded multi sets. Let $W = \{w_1, w_2, w_3, \ldots\}, V_1 = \{v_1, v_3, v_5, \ldots\}$ and $V_2 = \{v_0, v_2, v_4, \ldots\}$ be infinite sets of variables. Let $\Sigma = \{\{\cdot\}, \cup\}$ be the signature of multi sets where $\{\cdot\}$ is unary set construction and $\cup$ is set union. To simplify notation, we assume that $\{\cdot\}$ is of arbitrary finite arity. Let $V_{\text{hfnws}}(W \cup V_1)$ denote the domain of hereditarily finite non-wellfounded multi sets over $W \cup V_1$.[11] Let $\Delta = \{a, g, h\}$ be the signature of the rational tree algebra $\mathfrak{R}^{\Delta}(\Delta, W \cup V_2)$ where a is a constant, g a unary and f a binary function symbol. The constraint problem we consider is

$$\Gamma = \{y \doteq g(y), z \doteq g(\{a, z\}), x = \{g(\{a, z\}), y, f(\{a, a\}, \{g(w), f(a, x)\})\}\}.$$

Since Γ is not in decomposed form, we first transform it into this form by means of variable abstraction to receive $\hat{\Gamma} = \Gamma_0^{\Sigma} \wedge \Gamma_0^{\Delta}$ where

$$\begin{aligned}\Gamma_0^{\Delta} \;=\; & \{y \doteq g(y), z \doteq g(x_{21}), x_{22} \doteq a, x_{23} \doteq g(x_{25}), x_{24} \doteq f(x_{26}, x_{27}), \\ & x_{28} \doteq a, x_{29} \doteq a, x_{32} \doteq a, x_{30} \doteq g(w), x_{31} \doteq f(a, x)\}\end{aligned}$$

and

$$\begin{aligned}\Gamma_0^{\Sigma} \;=\; & \{x_{21} \doteq \{x_{22}, z\}, x \doteq \{x_{23}, y, x_{24}\}, x_{25} \doteq \{x_{28}, z\}, \\ & x_{26} \doteq \{x_{29}, x_{32}\}, x_{27} \doteq \{x_{30}, x_{31}\}\}.\end{aligned}$$

In Step 1 of the algorithm, we choose the following partion of the variables (where the bold variable is the representant of the class):

$$[\mathbf{x_{22}}, x_{28}, x_{29}, x_{32}], [\mathbf{x_{21}}, x_{25}], [x_{23}, \mathbf{z}], [\mathbf{y}], [\mathbf{x}], [\mathbf{x_{24}}], [\mathbf{x_{26}}], [\mathbf{x_{27}}], [\mathbf{x_{30}}], [\mathbf{x_{31}}].$$

After this identification, the problem looks as follows

$$\begin{aligned}\Gamma_1^{\Delta} \;=\; & \{y \doteq g(y), z \doteq g(x_{21}), x_{22} \doteq a, x_{24} \doteq f(x_{26}, x_{27}), \\ & x_{30} \doteq g(w), x_{31} \doteq f(a, x)\}\end{aligned}$$

and

$$\Gamma_1^{\Sigma} = \{x_{21} \doteq \{x_{22}, z\}, x \doteq \{z, y, x_{24}\}, x_{26} \doteq \{x_{22}, x_{22}\}, x_{27} \doteq \{x_{30}, x_{31}\}\}.$$

In Step 2, we choose the atom/non-atom declaration $(\{y, z, x_{22}, x_{24}, x_{30}, x_{31}\}, \{x, x_{21}, x_{26}, x_{27}\})$.

Now, the constraint problem with A/N declaration $\langle \Gamma_1^{\Delta}, \{x, x_{21}, x_{26}, x_{27}\}, \{y, z, x_{22}, x_{24}, x_{30}, x_{31}\}\rangle$ clearly has a solution in $\mathfrak{R}^{\Delta}(\Delta, W \cup V_2)$:

$$\begin{array}{rlrl} y & \mapsto gggg\ldots & x_{22} & \mapsto a \\ z & \mapsto g(v_2) & x_{21} & \mapsto v_2 \\ x_{24} & \mapsto f(v_4, v_6) & x_{26} & \mapsto v_4 \\ x_{30} & \mapsto g(w_1) & x_{27} & \mapsto v_6 \\ x_{31} & \mapsto f(a, v_8) & x & \mapsto v_8 \\ w & \mapsto w_1 & & \end{array}$$

[11] See Examples 3.2.17.

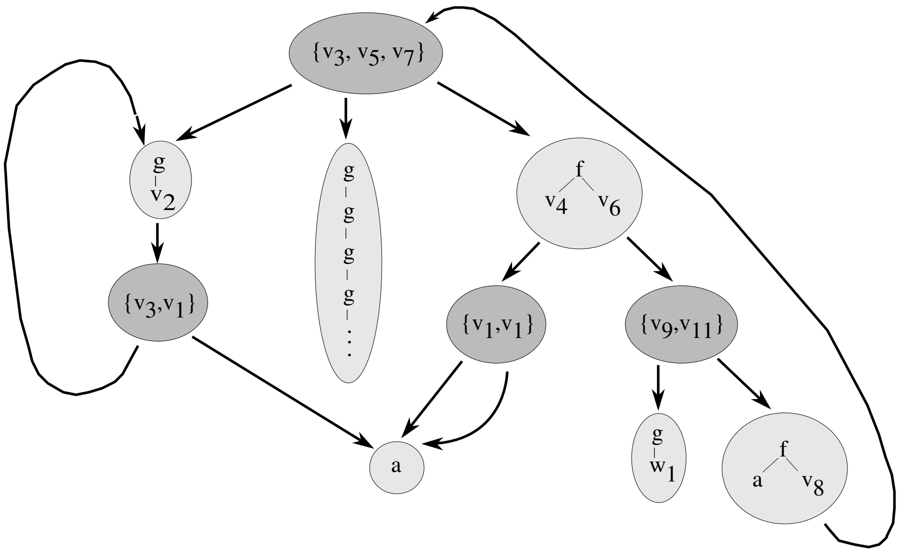
This is a solution of Γ_1^Δ, and all elements in $\{x, x_{21}, x_{26}, x_{27}\}$ are mapped to different atoms in $W \cup V_2$, while all variables in $\{y, z, x_{22}, x_{24}, x_{30}, x_{31}\}$ are mapped to non-atoms.

The constraint problem with A/N declaration $\langle \Gamma_1^\Sigma, \{y, z, x_{22}, x_{24}, x_{30}, x_{31}\}, \{x, x_{21}, x_{26}, x_{27}\}\rangle$ has a solution in $V_{\mathrm{hfnws}}(W \cup V_1)$:

$$\begin{array}{rlrl} x_{21} & \mapsto \{v_1, v_3\} & x & \mapsto \{v_3, v_5, v_7\} \\ x_{26} & \mapsto \{v_1, v_1\} & x_{27} & \mapsto \{v_9, v_{11}\} \\ x_{22} & \mapsto v_1 & z & \mapsto v_3 \\ y & \mapsto v_5 & x_{24} & \mapsto v_7 \\ x_{30} & \mapsto v_9 & x_{31} & \mapsto v_{11} \end{array}$$

This solves Γ_1^Σ, and the variables $\{y, z, x_{22}, x_{24}, x_{30}, x_{31}\}$ are mapped to different atoms, while the variables $\{x, x_{21}, x_{26}, x_{27}\}$ are mapped to non-atoms.

Hence by Theorem 5.5.3, there is a solution of Γ. The following figure shows how this solution looks like.



5.5.3 Proof of Theorem 5.5.5

In order to proof Theorem 5.5.5 we shall use the following variant of Algorithm 5.5.8.

Algorithm 5.5.12 The input constraint Γ, and Steps 1 and 2, remain as above. The output of Algorithm 5.5.12 consists of the two positive universal-existential sentences

$$\sigma = \forall \vec{u} \exists \vec{w} \exists \vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma$$

and

$$\delta = \forall \vec{w} \exists \vec{u} \exists \vec{v}_{1,\Delta}\ \Gamma_1^\Delta$$

where $\vec{u}$ ($\vec{w}$) represent the variables in U (resp. W), $\vec{v}_{1,\Sigma}$ represents the non-shared variables in Γ_1^Σ, and $\vec{v}_{1,\Delta}$ represents the non-shared variables in Γ_1^Δ.

Proposition 5.5.13 *The input formula Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$ if and only if there exists an output pair (σ, δ) of Algorithm 5.5.12 such that $\mathfrak{A}^\Sigma \models \sigma$ and $\mathfrak{B}^\Delta \models \delta$.*

Theorem 5.5.5 is an immediate consequence. In order to prove Proposition 5.5.13 we shall first show that Algorithm 5.5.12 is sound. As above we shall assume that the two components $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ have the form $(\mathfrak{A}^\Sigma, X, \mathcal{M})$ and $(\mathfrak{B}^\Delta, Y, \mathcal{N})$ respectively.

Lemma 5.5.14 *If, for some output pair (σ, δ) of Algorithm 5.5.12, $\mathfrak{A}^\Sigma \models \sigma$ and $\mathfrak{B}^\Delta \models \delta$, then Γ is solvable in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$.*

Proof. Assume that $\mathfrak{A}^\Sigma \models \forall\vec{u}\exists\vec{w}\exists\vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma$ and $\mathfrak{B}^\Delta \models \forall\vec{w}\exists\vec{u}\exists\vec{v}_{1,\Delta}\ \Gamma_1^\Delta$. Let $\vec{u} = u_1, \ldots, u_m$, let $\vec{w} = w_1, \ldots, w_n$. For each variable u_i we select a distinct atom $x_i \in X$ of A $(1 \leq i \leq m)$, and for each variable w_j we select a distinct atom $y_j \in Y$ of B $(1 \leq j \leq n)$. Then there are elements $a_1, \ldots, a_n \in A$ and $b_1, \ldots, b_m \in B$ such that

$$\begin{aligned} \mathfrak{A}^\Sigma &\models \exists\vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma(\vec{u}/\vec{x}, \vec{w}/\vec{a}) \\ \mathfrak{B}^\Delta &\models \exists\vec{v}_{1,\Delta}\ \Gamma_1^\Delta(\vec{u}/\vec{b}, \vec{w}/\vec{y}). \end{aligned}$$

We distinguish two cases.

First case: $x_i \neq a_j$ and $b_i \neq y_j$, for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Since $\mathfrak{A}^\Sigma$ is non-trivial, we may choose an endomorphism $m_1 \in \mathcal{M}$ that maps all atoms in the set $\{a_1, \ldots, a_n\}$ to a non-atomic element $a \in A$ and fixes all other atoms. In particular, m_1 leaves the atoms $x_1, \ldots, x_m$ fixed, by assumption. Since $\mathfrak{A}^\Sigma$ is non-collapsing, all elements in the set $\{m_1(a_1), \ldots, m_1(a_n)\}$ are non-atomic. Since Γ_1^Σ is a positive formula we have

$$\mathfrak{A}^\Sigma \models \exists\vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma(\vec{u}/\vec{x}, \vec{w}/\vec{m}_1(a)),$$

by Lemma 5.2.1. It follows that the Σ-constraint with A/N declaration, (Γ_1^Σ, U, W), has a solution in $\mathfrak{A}^\Sigma$.

Symmetrically we may choose an endomorphism $n_1 \in \mathcal{N}$ such that all elements in $\{n_1(b_1), \ldots, n_1(b_m)\}$ are non-atomic and

$$\mathfrak{B}^\Delta \models \exists\vec{v}_{1,\Delta}\ \Gamma_1^\Delta(\vec{u}/n_1(\vec{b}), \vec{w}/y).$$

It follows that the Δ-constraint with A/N declaration, (Γ_1^Δ, W, U), has a solution in $\mathfrak{B}^\Delta$. Now Lemma 5.5.10 shows that the input formula Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$.

Second case: Without loss of generality, $x_i = a_j$, for some $1 \leq i \leq m$ and $1 \leq j \leq n$. We consider the new formula $\Gamma'_{1,\Sigma}$ $(\Gamma'_{1,\Delta})$ that is obtained by replacing all occurrences of w_i in Γ_1^Σ (resp. Γ_1^Δ) by u_j. Consider the pair with the formulae $\sigma' = \forall\vec{u}\exists\vec{w}'\exists\vec{v}_{1,\Sigma}\ \Gamma'_{1,\Sigma}$ and $\delta' = \forall\vec{w}'\exists\vec{u}\exists\vec{v}_{1,\Delta}\ \Gamma'_{1,\Delta}$, where the sequence $\vec{w}'$ is obtained from $\vec{w}$ by removing w_i. Obviously, (σ', δ') is again an output pair of Algorithm 5.5.12. We claim that $\mathfrak{A}^\Sigma \models \sigma'$ and $\mathfrak{B}^\Delta \models \delta'$.

We have

$$\mathfrak{A}^\Sigma \models \exists \vec{v}_{1,\Sigma}\ \Gamma'_{1,\Sigma}(\vec{u}/\vec{x}, \vec{w}'/\vec{a}'),$$

where $\vec{a}'$ denotes the sequence $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n$. Since X is an $\mathcal{M}$-atom set, for each sequence $\vec{c} = c_1, \ldots, c_m$ of elements of A there exists an endomorphism $m_2 \in M$ such that $m_2(x_i) = c_i$, for $1 \leq i \leq m$. Now Lemma 5.2.1 shows that $\mathfrak{A}^\Sigma \models \sigma'$.

Since $(\mathfrak{B}^\Delta, Y, \mathcal{N})$ is rational, there exists an endomorphism $n_2 \in \mathcal{N}$ that leaves all atoms but y_i fixed such that $n_2(y_i) = n_2(b_j)$. By Lemma 5.2.1,

$$\mathfrak{B}^\Delta \models \exists \vec{v}_{1,\Delta}\ \Gamma'_{1,\Delta}(\vec{u}/n_2(\vec{b}), \vec{w}'/\vec{y}'),$$

where the sequence $\vec{y}'$ is obtained from $\vec{y}$ by removing y_i. Since the elements in the sequence $\vec{y}'$ are distinct atoms it follows as above that $\mathfrak{B}^\Delta \models \delta'$.

In this second case we have seen that we can construct a new output pair (σ', δ') of Algorithm 5.5.12 such that $\mathfrak{A}^\Sigma \models \sigma'$ and $\mathfrak{B}^\Delta \models \delta'$. Moreover, the number of variables in (σ', δ') is strictly smaller than the number of variables in (σ, δ). We may now use the same subcase analysis as above, replacing (σ, δ) by (σ', δ'), and iterate this contraction of formulae, if necessary. After a finite number of steps we reach an output pair that satisfies all the assumptions that we made for (σ, δ) in the first subcase. As we have seen, this shows that the input formula Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$. ∎

As the last step, we show completeness of Algorithm 5.5.12.

Lemma 5.5.15 *If the input constraint Γ has a solution in $\mathfrak{A}^\Sigma \odot \mathfrak{B}^\Delta$, then there exists an output pair (σ, δ) of Algorithm 5.5.12 such that $\mathfrak{A}^\Sigma \models \sigma$ and $\mathfrak{B}^\Delta \models \delta$.*

Proof. Lemma 5.5.11 shows that Algorithm 5.5.8 has an output pair $(\langle \Gamma_1^\Sigma, U, W\rangle, \langle \Gamma_1^\Delta, W, U\rangle)$ such that $\langle \Gamma_1^\Sigma, U, W\rangle$ has a solution in $\mathfrak{A}^\Sigma$ and $\langle \Gamma_1^\Delta, W, U\rangle$ has a solution in $\mathfrak{B}^\Delta$. In $\mathfrak{A}^\Sigma$, variables of U are interpreted as distinct atoms in X under the given solution. Lemma 5.2.2 shows that $\mathfrak{A}^\Sigma \models \forall \vec{u} \exists \vec{w} \exists \vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma$. In $\mathfrak{B}^\Delta$, variables of W are interpreted as distinct atoms in Y under the given solution. By Lemma 5.2.2, $\mathfrak{B}^\Delta \models \forall \vec{w} \exists \vec{u} \exists \vec{v}_{1,\Delta}\ \Gamma_1^\Delta$. This shows that the sentences $\sigma := \forall \vec{u} \exists \vec{w} \exists \vec{v}_{1,\Sigma}\ \Gamma_1^\Sigma$ and $\delta := \forall \vec{w} \exists \vec{u} \exists \vec{v}_{1,\Delta}\ \Gamma_1^\Delta$ of the corresponding output pair (σ, δ) of Algorithm 5.5.12 are valid in $\mathfrak{A}^\Sigma$ and $\mathfrak{B}^\Delta$ respectively. ∎

5.6 Conclusion

In this chapter, we introduced rational amalgamation, a general methodology for combining constraint systems. The concept of a braid presents a way to interweave arbitrary elements of two quasi-free structures, in that a particular set of atoms, the open atoms in the stabiliser of an element of one structure are used as pointers to elements in the other structure. The amalgamation is rational in the sense that one may interweave only a finite number of elements in one and the same braid, but if one follows a pointer chain in a braid, cycles are permitted. We showed that for each braid there exists a standard normal form. And the set of braids in normal form is consequently used as the carrier for rational amalgamation. Together with the algebraic structure defined on top of it, this rational amalgam is the combined solution domain. We showed that the reduct of the rational amalgam to a component signature is isomorphic to the component structure proving this way conservativeness of the combination. We also saw that the free amalgamated product is – modulo isomorphism – a substructure of the rational amalgam. And the rational amalgamation of two rational tree algebras is isomorphic to the rational tree algebra over the union of the signatures. We presented a decomposition algorithm to reduce the solving of mixed constraints in the rational amalgam to solving pure constraints in the components. The algorithm is similar to the decomposition algorithm for free amalgamation, but contains only two non-deterministic steps. We proved that solvability of mixed constraints over the joint signature in the rational amalgam is decidable, if solvability of pure constraints with so-called atom-non-atom declarations is decidable in both components. For the subclass of so-called rational quasi-free structures, the result can be described in purely logical terms. The existential positive theory of the rational amalgam is decidable, provided the universal-existential positive theory is decidable in the component structures.

The present chapter, in connection with the discussion of free amalgamation in Chapter 3 and [10, 15], seems to suggest a new view of the problem of combining solution domains and constraint solvers. There is now strong evidence that the situation considered in Chapter 3 and here – the construction of "mixed" elements of a combined domain, given the "pure" elements of two component structures as construction units – is quite similar to the process of building the elements of a single structure, given the symbols of a fixed signature as construction units. We are confident that this analogy will help to isolate the most important methods for combining structures over disjoint signatures, and to understand the relationship and the differences between different amalgamation constructions.

When we compose elements, given the symbols of a fixed signature, three different structures may be obtained in a direct way, depending on the composition principle, namely the free term algebra, the algebra of rational trees, and the algebra of infinite trees. The privileged role of these three algebras, and the rich amount of interesting relationships between them, are now well-understood (e.g., [3, 35, 36, 70]). We believe that free amalgamation, rational amalgamation and a further construction called "infinite amalgamation" (still to be investigated) reflect this role on the higher level of amalgamation constructions. Many of the results that we have obtained for free and rational amalgamation can be interpreted in this sense:

- The universality-property of the free amalgamated product reflects the status of the free term algebra as the absolutely free Σ-algebra.
- We saw that the free amalgamated product is always a substructure of

the rational amalgamated product. This reflects the fact that the free term algebra is always a substructure of the algebra of rational trees.

- It is well-known that the unification algorithm for the algebra of rational trees can be considered as the variant of the unification algorithm for the free term algebra where we omit the occur-check. Similarly, the decomposition scheme for rational amalgamation as given here is – essentially – the decomposition scheme for free amalgamation where we omit the "interstructural" occur-check that is provided by the choice of a linear ordering in the latter scheme.

We would not be surprised if more principles, techniques and theorems, well-known on the level of tree constructions, could be lifted to the level of combining structures. Our experience with rational amalgamation seems to indicate that this is a difficult, but promising line of research if we want to understand the scale of possibilities, and the limitations for combining solution domains and constraint solvers.

Chapter 6

Negation in Combining Constraint Systems

6.1 Introduction

The need of negation in constraint solving is so obvious that it hardly requires an explanation. Formulation of many problems just naturally involve some type of negation. Negation is also required for implication and constraint entailment, and both are used heavily in actual implementations of constraint solvers when reducing sets of constraints. In previous chapters however, the constraints we had been looking at were exclusively positive; there was no negation involved so far. The underlying hidden reason thereof was given very early in Lemma 2.1.1 that states that surjective homomorphisms preserve *positive* formulae. This lemma figures prominently in many correctness proofs, even if we did not always explicitly cite it. Hence one could think that handling negative constraints in combining constraint solvers is impossible. But already back in 1993, F. Baader and K. U. Schulz [6] showed that the combination technique for deciding combined equational unification problems they had published the year ago [5] can be extended to the combination of *dis*unification problems. Thus the question naturally arises, whether the results obtained for combining equational disunification problems can be extended to the more general case of solving positive and negative constrains in combinations of quasi-free structures. The correctness proof given in [6] (see also [9]) uses complicated rewriting methods and is very technical in nature. Even if one is able to manually check the individual deduction steps in the proof, one cannot gain any insight for why the combination theorem is correct. Quasi-free structures on the other hand are defined algebraically, rewriting methods are thus not applicable. This forced us to have a fresh look at the problem.

This chapter is divided into two parts. The first one covers the solving of positive and negative mixed constraints in the combination of constraint systems. When looking at negative constraints, one often distinguishes two cases, and we do this here, too: A general case in which the solutions of problems may contain variables or atoms and a ground case where all solution elements must be ground. For the general case, we will show that the existential theory of mixed constraints is decidable in the free amalgamated product, if conjunctions of pure constraints with linear constant restrictions are decidable in the component structures. For the ground case we present the following result. The existential theory of the ground substructure of the free amalgamated product is decidable, if the existence of so-called restrictive solutions of pure constraints with linear constant restrictions is decidable in the components. In the second half of this part, we try to see if these results can be strengthened or expressed in less technical terms, i.e., without the notion of a linear constant restriction. We will also show that in general, these results cannot be carried over rational amalgamation, the other principled method of combination.

The second part of this chapter deals with the independence property of negative constraints. A constraint system is said to have the independence property, if the solvability of a conjunction of positive constraints and a conjunction of negative constraints can be reduced to solving each single negative conjunct together with the conjunction of positive constraints. In other words, if for some negative constraints, each of them is solvable separately, then their conjunction is solvable. This property plays an important role in real world constraint solvers that cope with negative constraints (see, e.g., [49]). Quite often, these constraint solvers can handle only positive constraints in an efficient manour. The independence property can then be used to reduce negative constraints in such solvers in the following way. First one solves the conjunction of the positive constraints to see if they have a solution at all. Then for each negative constraint, one constructs the dual positive one and tries to solve the conjunction of the positive constraints enlarged by the dual. If this is not solvable, then the negative constraint can be solved. If the solution is identical to the one for the constraints without the dual, the negative constraint cannot be solved. If the solution is more specific than that for the constraints without the dual, then the negative constraint is solvable. The independence property ensures that processing each of the negative constraints separately this way still provides a decision procedure for the whole conjunction of negative constraints.

The aim of this second part is to give a modularity result for the independence property, that is to state under which circumstance the free amalgamated product owns the independence property provided the component structures do so. We will finally show: The free amalgamation of two unitary regular and non-collapsing quasi-free structures has the independence property. To get there is quite a way. First we look at a particular subclass of quasi-free structures, namely equational theories and unification, since there is not yet any discussion of this topic for these prototypical quasi-free structures. We find that unitary equational theories have the independence property, while finitary do not. We also present a special class of equational theories, the monoidal or commutative theories and prove that the whole class has the independence property showing thereby that there are non-unitary theories that have the independence property. Then we give a first modularity result. Based on work by K. U. Schulz [94], who shows that there exists a deterministic combination algorithm for the combination of unitary regular collapse-free equational theories, we demonstrate that the combination of unitary regular collapse-free equational theories is again unitary and hence has the independence property. We proceed by lifting these results to quasi-free structures. We prove that unitary quasi-free structures have the independence property. And generalising the proofs for combining equation theories, we show that the combination of unitary regular and non-collapsing quasi-free structures is again unitary.

In this chapter, a constraint problem is a conjunction of literals, i.e., of atomic and negated atomic formulae.

Part I

Combination of Constraint Solvers

6.2 Free Amalgamation of Negative Constraints: The General Case

In this section, we show that solving mixed constraints in the free amalgamation can be reduced to solving pure constraints in the components, even if the constraints contain negations. The type of problems we have to solve in the components are constraint problems with linear constant restrictions. Problems with linear constant restrictions are defined in 4.2.2. Remember that a linear constant restriction $L = (Lab, <_L)$ consists of a labelling and a linear order of the shared variables. A solution of a constraint problem with linear constant restriction (Γ^Σ, L) is a solution of Γ^Σ such that each variable in the domain of Lab that is not assigned to Σ must be mapped to an atom, and this atom must not appear in the stabiliser of any element that is a solution of a variable which is strictly smaller according to the order $<_L$. Now we can present the main theorems of the first part.

Theorem 6.2.1 *Let $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ be two quasi-free structures over disjoint signatures. The solvability of mixed $\Sigma_1 \cup \Sigma_2$ constraint problems, i.e., conjunctions of literals, in the free amalgamated product $\mathfrak{A}_1^{\Sigma_1} \otimes A_2^{\Sigma_2}$ is decidable, if solvability of constraint problems with linear constant restrictions is decidable in both components $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$.*

Since existential quantifiers distribute over disjunctions, we can strengthen our result a little bit.

Theorem 6.2.2 *Let $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ be two quasi-free structures over disjoint signatures. The existential theory (and the universal theory) of the free* *amalgamated product $\mathfrak{A}_1^{\Sigma_1} \otimes A_2^{\Sigma_2}$ is decidable, if solvability of constraint problems with linear constant restrictions is decidable in both components $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$.*

One application of this theorem is the following

Corollary 6.2.3 *The existential theory of the free amalgamated product of the following signature-disjoint component quasi-free structures is decidable:*

- *free algebras defined by these equational theories*
  - *the empty theory (syntactic unification),*
  - *the theory A of an associative function symbol,*
  - *the theory AC of an associative-commutative function symbol,*
  - *the theory ACI of an associative-commutative-idempotent function symbol,*
- *rational tree algebras,*
- *feature structures*
- *well-founded sets, multi-sets and lists.*

Proof. Decidability of disunification problems with linear constant restrictions in the free theory and the theories A, AC, and ACI was proven by F. Baader and K. U. Schulz in [9]. Decidability of disunification problems with linear constant restrictions in rational tree algebras follows directly from the result for the free theory. Decidability of constraint problems with linear constant restrictions in feature structures is a simple corollary of the work by G. Smolka and R. Treinen [104] and the work by P. van Roy, M. Mehl, and R. Scheidhauer [114]. Finally, decidability of constraint problems with linear constant restrictions in sets, multi-sets and lists can be reduced to corresponding disunification problems in the theories ACI, AC, and A respectively. ∎

The Decomposition Algorithm

As stated above, we would like to solve mixed positive and negative constraints in the free amalgamated product of two quasi-free by reduction to solving pure constraints in the components. Hence we need a decomposition algorithm. The one that follows below is very similar to the algorithm in Section 3.4.1 designed for decomposing positive constraints. The main extension deals with disequations. Disequations should only occur between variables. Thus if $s \not\doteq t$ is a disequation between pure terms s, t, we replace that disequation by two new equations $x_1 \doteq s, x_2 \doteq t$ and the disequation $x_1 \not\doteq x_2$. Negated predicates different from equality are handled in a straight forward manour in that alien subterms are removed by variable abstraction. Additionally, we ensure that no two representatives after variable identification receive the same solution value by introducing a disequation $r_1 \not\doteq r_2$ for each two different representatives r_1, r_2.

The decomposition algorithm is almost the same as the one given by F. Baader & K. U. Schulz in [9] for disunification problems, it is minimally extended to handle negated predicates different from disequality.

Algorithm 6.2.4 The input Γ_0 is a constraint problem over signature $\Sigma := \Sigma_1 \cup \Sigma_2$.

Step 1: variable abstraction
Alien subterms are successively replaced by new variables until all literals and terms occurring in the system are pure.

Step 2: split non-variable disequations and non-pure equations
Each disequation of the form $s \neq t$ (where s or t is not a variable) is replaced by two equations $x \doteq s, y \doteq t$ and a disequation $x \neq y$, where x and y are always new variables. Each non-pure equation of the form $s \doteq t$ is replaced by two equations $x \doteq s, x \doteq t$, where x is always a new variable.

Step 3: variable identification
Non-deterministically choose a partition of the set of all shared variables of the constraint problem such that whenever the constraint problem contains a disequation $x \neq y$ then x and y belong to different classes of the partition. The variables of each class are identified by choosing a representative for each class and replacing each variable by its representative. In addition, add the disequation $x \neq y$ for each pair x, y of distinct representatives to the system, if this disequation is not already present.

Step 4: variable labelling and ordering
For a given system, choose a mapping Lab from the set of variables into the set of theory labels $\{\Sigma_1, \Sigma_2\}$ and a strict linear order $<$ on the variables. This pair $L = (Lab, <_L)$ gives rise to a linear constant restriction.

Step 5: split systems
A given system Γ_4 is split into two systems $\Gamma_{5,1} \cup \Gamma_{5,2}$ where $\Gamma_{5,1}$ contains only 1-(dis)equations and 1-literals and $\Gamma_{5,2}$ contains only 2-(dis)equations and 2-literals. Additionally, the system $\Gamma_{5,i}$ must contain all disequations $x \neq y$ where x or y has label Σ_i. This means that disequations between variables of distinct labels are put into both subsystems. The subsystems can now be considered as constraint problems with linear constant restriction.

Soundness

In this subsection, we prove that the above algorithm is sound.

Proposition 6.2.5 *The input problem Γ_0 is solvable, if there exists an output pair $(\Gamma_{5,1}, \Gamma_{5,2})$ and a linear constant restriction L such that $(\Gamma_{5,1}, L)$ is solvable in $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\Gamma_{5,2}, L)$ is solvable in $(\mathfrak{A}_2^{\Sigma_2}, X)$.*

Before we can present the proof, we need some technical preparations and lemmata.

Lemma 6.2.6 *Let* $m : (\mathfrak{A}^{\Sigma}, X) \to (\mathfrak{B}^{\Sigma}, Y)$ *be a qf-isomorphism.*
If $\mathrm{Stab}^{\mathfrak{A}}(a) = \{x_1, \ldots, x_n\}$, *then* $\mathrm{Stab}^{\mathfrak{B}}(m(a)) = \{m(x_1), \ldots, m(x_n)\}$.

Proof. Let $\mathrm{Stab}^{\mathfrak{A}}(a) = \{x_1, \ldots, x_n\}$. Firstly, we show that $\mathrm{Stab}^{\mathfrak{B}}(m(a)) \subseteq \{m(x_1), \ldots, m(x_n)\}$. Let $\nu_1, \nu_2 \in End_{\mathfrak{B}}^{\Sigma}$ such that ν_1 and ν_2 coincide on $\{m(x_1), \ldots, m(x_n)\}$. Since m and its inverse are qf-isomorphisms, the $\mathfrak{A}$-endomorphisms $m^{-1}\nu_1 m$ and $m^{-1}\nu_2 m$ coincide on $\{x_1, \ldots, x_n\}$. Because a is stabilised by $\{x_1, \ldots, x_n\}$, also $m^{-1}\nu_1 m =_{\{a\}} m^{-1}\nu_2 m$ which is the same as $m^{-1}\nu_1 =_{\{m(a)\}} m^{-1}\nu_2$. Since m^{-1} is a qf-isomorphism, $\nu_1 =_{\{m(a)\}} \nu_2$.
Now suppose $\mathrm{Stab}^{\mathfrak{B}}(m(a))$ is a proper subset of $\{m(x_1), \ldots, m(x_n)\}$. Then $a = m^{-1}(m(a))$ is stabilised by a proper subset of $m^{-1}(\{m(x_1), \ldots, m(x_n)\}) = \{x_1, \ldots, x_n\}$ which contradicts the choice of $\{x_1, \ldots, x_n\}$. ∎

Lemma 6.2.7 *Let* L *be a linear constant restriction and* σ_i *be a solution of* $(\Gamma_{5,i}, L)$ *and* x, y *two distinct variables in* $\Gamma_{5,i}$ *(where* $i = 1, 2$*). Then* $\sigma_i(x) \neq \sigma_i(y)$, *i.e.,* σ_i *is injective.*

Proof. If $x, y \in Var(\Gamma_{5,i})$ and $x \neq y$, then x and y must be different representatives of different classes after variable identification in Step 3. By definition of Step 3, we add the disequation $x \not\doteq y$. Therefore $x \not\doteq y \in \Gamma_{5,i}$. Since σ_i solves $\Gamma_{5,i}$, we have $\sigma_i(x) \neq \sigma_i(y)$. ∎

Lemma 6.2.8 *Let* $(\mathfrak{A}^{\Sigma}, X)$ *be a quasi-free structure and* σ *be a solution of the constraint problem with linear constant restriction* $(\Gamma, Lab, <)$. *Let* $\mu : (\mathfrak{A}^{\Sigma}, X) \to (\mathfrak{B}^{\Sigma}, Y)$ *be a qf-isomorphism. Then* $\mu \circ \sigma$ *is a solution of* $(\Gamma, Lab, <)$ *in* $(\mathfrak{B}^{\Sigma}, Y)$, *i.e., qf-isomorphisms preserve solutions.*

Proof. Since μ is an isomorphism, $\mu\sigma$ is a solution of Γ. We have to show that respecting linear constant restrictions is preserved under qf-isomorphisms.

Let $Lab(x) = \Delta$ and $\Delta \neq \Sigma$.
Then $\sigma(x) \in X$, because σ solves the linear constant restriction.
Then $\mu\sigma(x) \in X$, because μ is a **qf**-isomorphism.

Let $Lab(x) = \Delta$, $Lab(y) = \Delta$, $\Delta \neq \Sigma$ and $x \neq y$.
Then $\sigma(x) \neq \sigma(y)$, because σ solves the linear constant restriction.
Then $\mu\sigma(x) \neq \mu\sigma(y)$, because μ is injective.

Let $x < y$, $Lab(x) = \Sigma$, $Lab(y) = \Delta$ and $\Delta \neq \Sigma$.
Then $\sigma(y) \notin \mathrm{Stab}^{\mathfrak{A}}(\sigma(x))$, because σ solves the linear constant restriction.
Then $\mu\sigma(y) \notin \mu(\mathrm{Stab}^{\mathfrak{B}}(\sigma(x))) = \mathrm{Stab}^{\mathfrak{B}}(\mu\sigma(x))$, because μ is injective where the equality is justified by Lemma 6.2.6. ∎

Definition 6.2.9 Let $(\mathfrak{C}_1^{\Sigma}, X)$ be the free amalgamated product of $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(A_2^{\Sigma_2}, X)$ and $h_{1,2}$ the qf-isomorphism between $(\mathfrak{C}_1^{\Sigma}, Z_1)$ and $(\mathfrak{C}_2^{\Sigma}, Z_2)$. The *shadow* of an element $a \in C_1$ is defined recursively:

$\mathrm{Sd}(a) := \{a\} \cup \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(a) \cup \{\mathrm{Sd}(h_{1,2}(x)) \mid x \in \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(a)\}$. Analogously, the shadow of $b \in C_2$ is defined as $\mathrm{Sd}(b) := \{b\} \cup \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(b) \cup \{\mathrm{Sd}(h_{2,1}(y)) \mid y \in \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(b)$.

An element $b \in \mathrm{Sd}(a)$ is called a *bottom element*, iff $\mathrm{Sd}(b) = \{b\}$. The set $\mathrm{Bottom}(\mathrm{Sd}(a))$ denotes the set of all bottom elements in the shadow of a.

Intuitively speaking, the shadow of an element is the element and "everything below" it where one descends via the ladder constructed by the stabilisers and the isomorphisms $h_{1,2}$ and $h_{2,1}$. So the shadow contains the element, its stabiliser, the fibre images of the stabiliser elements, their stabilisers, the fibre images thereof and so on, everything pending below the element via stabilisers and fibre images.

Lemma 6.2.10 *The shadow of every element is finite. Its bottom elements are elements with empty stabiliser or atoms in X.*

Proof. For all elements $a \in C_1 \cup C_2$ holds $a \in \mathrm{Sd}(a)$. If $x \in X$, then x is fibred with itself, and $\mathrm{Stab}(x) = \{x\}$. Thus $\mathrm{Sd}(x) = \{x\}$. If $b \in C_1 \cup C_2$ is such that $\mathrm{Stab}(b) = \emptyset$, then clearly $sd(b) = \{b\}$. Any other element has a non-empty stabiliser which is not fibred with itself, hence the shadow contains at least the stabilisers and their fibre images, and the element is not a bottom element.

The shadow of an element a can be seen as a tree with the element as its root. Since each element has a finite stabiliser, the tree is finitely branching. A branch in the tree is a sequence $(a_{1,}, a_2, a_3, \ldots)$ such that $a_1 = a$, for all $i, j \geq 1 : i \neq j$ implies $a_i \neq a_j$, and for each $i \geq 1$ either a_{i+1} is the fibre image of a_i or $a_{i+1} \in \mathrm{Stab}(a_i)$. Now remember the definition of the height of an element (Def. 3.3.6 on page 34). Clearly, if a_{i+1} is the fibre image of a_i, then $\mathit{height}(a_{i+1}) = \mathit{height}(a_i)$. And if $a_{i+1} \in \mathrm{Stab}(a_i)$, then $\mathit{height}(a_{i+1}) < \mathit{height}(a_i)$, because $a_{i+1} \neq a_i$ and therefore a_i is non-atomic. Hence, for each $i \geq 1 : \mathit{height}(a_{i+2}) < \mathit{height}(a_i)$. The height stepwise decreases in a branch. But since the height of a, the root, is finite, the height can decrease only a finite number of steps. Therefore, each branch must be finite. ∎

Proof of the soundness proposition (Proposition 6.2.5).
Let $\mathfrak{C}_1^{\Sigma}$ be the free amalgamated product of $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(A_2^{\Sigma_2}, X)$, and let σ_1 be the solution of $(\Gamma_{5,1}, L)$ in $(\mathfrak{A}_1^{\Sigma_1}, X)$ and σ_2 the solution of $(\Gamma_{5,2}, L)$ in $(\mathfrak{A}_2^{\Sigma_2}, X)$ where $L = (Lab, <_L)$. Let $h_{1,2} : \mathfrak{C}_1^{\Sigma} \to \mathfrak{C}_2^{\Sigma}$ be the qf-isomorphism between $\mathfrak{C}_1^{\Sigma}$ and $\mathfrak{C}_2^{\Sigma}$.

Let $h_1 : \mathfrak{A}_1^{\Sigma_1} \to \mathfrak{C}_1^{\Sigma_1}$ be the qf-isomorphism between $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{C}_1^{\Sigma_1}$. Then $h_1\sigma_1$ solves $(\Gamma_{5,1}, L)$ in $\mathfrak{C}_1^{\Sigma_1}$ ($\mathfrak{C}_1^{\Sigma}$) by Lemma 6.2.8.

Let $h_2 : \mathfrak{A}_2^{\Sigma_2} \to \mathfrak{C}_2^{\Sigma_2}$ be the qf-isomorphism between $\mathfrak{A}_2^{\Sigma_2}$ and $\mathfrak{C}_2^{\Sigma_2}$. Then $h_2\sigma_2$ solves $(\Gamma_{5,2}, L)$ in $\mathfrak{C}_2^{\Sigma_2}$ by Lemma 6.2.8.

Let $V = \mathit{Var}(\Gamma_{5,1}) \cup \mathit{Var}(\Gamma_{5,2})$ be the set of all variables. If we had $h_{1,2}h_1\sigma_1(v) = h_2\sigma_2(v)$ for all $v \in V$, then we would be done, because then, obviously, $h_1\sigma_1$

would be a solution of $\Gamma_{5,2}$ in $\mathfrak{C}_1^{\Sigma}$. Since the solutions σ_1 and σ_2 are found independently of each other, we cannot expect this condition to hold a priory. And we do not want to restrict the type of admitted solutions in the components. So the task is to show that the given solutions $h_1\sigma_1$ and $h_2\sigma_2$ can be transformed by means of automorphisms in such a way that finally the value of a variable $v \in V$ under $h_1\sigma_1$ is the fibre image of the value under $h_2\sigma_2$. We call this the fibring condition for v. The use of automorphisms is required to handle negative constraints: While endomorphisms preserve only the validity of positive formulae, automorphisms preserve validity of arbitrary formulae.

The generalised linear constant restriction L contains a linear order $<_L$ of the variables V. Let $v_1, v_2, v_3, \ldots, v_n$ be the enumeration of the variables alongside the order. By induction on this enumeration we prove that there exist solutions $l_{i,1}$ of $(\Gamma_{5,1}, L)$ and $l_{i,2}$ of $(\Gamma_{5,2}, L)$ such that for all variables v_j with $j \leq i$ we have $h_{1,2} \circ l_{i,1}(v_j) = l_{i,2}(v_j)$, that is the first i variables fulfil the fibring condition. But this simple statement alone is too weak. In an induction step, we may need to apply an automorphism exchanging two atoms in order to establish the fibring condition for the current variable. The difficult part consists in showing that this automorphism does not dissolve the fibring conditions for variables already handled. This is where the shadows of elements well be needed. We assume that every variable occurring in the shadows of the first i variables already fulfils the fibring condition. And we show that the automorphisms are the identity on the shadows of the first i variables and hence do not dissolve an already established fibring condition.

Before we start the induction, we sort out a simple subcase. Let X be partitioned as $X_0 \uplus X_1 \uplus X_2$ where X_1 and X_2 are infinite, X_0 has at least n elements, and $X_0 \cap (\mathrm{Stab}^{\mathfrak{C}_1}(h_1\sigma_1(V)) \cup \mathrm{Stab}^{\mathfrak{C}_2}(h_2\sigma_2(V))) = \emptyset$. Define $V_Z := \{v \in V \mid h_1\sigma_1(v) \in Z_1 \text{ and } h_2\sigma_2(v) \in Z_2\}$ and $V_{\bar{Z}} := V \setminus V_Z$. In other words, V_Z is the set of all those variables which are mapped to an atom in $\mathfrak{C}_1^{\Sigma_1}$ as well as $\mathfrak{C}_2^{\Sigma_2}$ and $V_{\bar{Z}}$ is its complement, i.e., one of the solution elements is a non-atom for each of these variables.

Define the following two qf-isomorphisms. For all variables $v \in V_Z$ choose a new atom $x \in X_0$ and mappings $p_1 : h_1\sigma_1(v) \longleftrightarrow x$[1] and $p_2 : h_2\sigma_2(v) \longleftrightarrow x$. Since σ_1 is injective (Lemma 6.2.7) and h_1 is a qf-isomorphism, all the $h_1\sigma_1(v)$'s are distinct, and the x's are distinct by choice. Thus p_1 is a permutation on Z_1. Therefore there is a unique extension to a qf-automorphism $\pi_1 : \mathfrak{C}_1 \to \mathfrak{C}_1$. By Lemma 6.2.8, $\pi_1 h_1\sigma_1$ solves $(\Gamma_{5,1}, L)$. Analogously, π_2 is the unique extension of the permutation p_2 on Z_2 to a qf-automorphism, and $\pi_2 h_2\sigma_2$ solves $(\Gamma_{5,2}, L)$. Furthermore, since every element $x \in X$ constitutes a 1-fibre and thus $h_{1,2}(x) = x$, we see that for every variable v such that both $\pi_1 h_1\sigma_1(v) \in Z_1$ and $\pi_2 h_2\sigma_2(v) \in Z_2$ holds $\pi_2 h_2\sigma_2(v) = h_{1,2}\pi_1 h_1\sigma_1(v)$. That is to say for $v \in V_Z$ the fibring condition holds.

We will now prove by induction over the variables $v_1, v_2, \ldots, v_n$ as given by the linear order $<_L$ that for $i = 0, \ldots, n$ the following 4 properties hold.

[1] $a \longleftrightarrow b$ denotes the permutation of a and b, i.e., $a \mapsto b$ and $b \mapsto a$.

1. There is an assignment $l_{i,1} : V \to C_1$ such that $l_{i,1}$ solves $(\Gamma_{5,1}, L)$ and there is an assignment $l_{i,2} : V \to C_2$ such that $l_{i,2}$ solves $(\Gamma_{5,2}, L)$.

2. For every variable $v \in V_Z$: $l_{i,2}(v) = h_{1,2} l_{i,1}(v) \in X_0$.

3. Define
$$S_i := \bigcup_{j \leq i} \mathrm{Sd}(l_{i,1}(v_j)) \cup \bigcup_{j \leq i} \mathrm{Sd}(l_{i,2}(v_j)).$$
For every variable v such that $l_{i,1}(v) \in S_i$ or $l_{i,2}(v) \in S_i$ holds $l_{i,2}(v) = h_{1,2} l_{i,1}(v)$.

4. $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}})) \subseteq S_i \cup X_1$, $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V_{\bar{Z}})) \subseteq S_i \cup X_2$.

Before we dive into the technicalities of the induction, a few comments are in place. The aim of the induction is to show that we can transform the given solutions $\pi_1 h_1 \sigma_1$ and $\pi_2 h_2 \sigma_2$ stepwise by means of automorphisms in such a way that finally for every variable $v \in V$ the fibring condition holds. The first property says we still have solutions for $(\Gamma_{5,1}, L)$ and $(\Gamma_{5,2}, L)$. For variables in V_Z, those variables that are assigned an atom in both solutions, the fibring condition is already established. Item (2) says we do not loose this property. In each step of the induction, we consider a particular variable v_i. For this v_i, we want to establish the fibring condition. Now, in one of the solutions, the variable is assigned to an atom; and there is another atom, the atom that the non-atom in the other solution is fibred to. So in order to establish the fibring condition, we want to apply the transposition that exchanges these two atoms. The difficult problem is, of course, to show that this transposition does not dissolve the fibring conditions of those variables we have already taken handled. That is what Properties (3) and (4) have to ensure. The set S_i is the union of all shadows of all solutions for variables up to v_i in the enumeration. We demand that for each such element in S_i that is the solution of any variable we can be sure that the fibring condition holds. And Property (4) gives us a strong control over the stabilisers of non-atom solutions. Either they are in S_i, which means we have already taken care of them. Or they are in X_1 (resp. X_2), which means they are harmless.

Induction Base:
Define firstly the following map $\hat{l}_{0,1} : Z_1 \to X_1$ by permuting every $z \in \mathrm{Stab}^{\mathfrak{C}_1}(\pi_1 h_1 \sigma_1(V_{\bar{Z}}))$ with a different atom in X_1. It extends uniquely to a qf-automorphism $\tilde{l}_{0,1}$ of $(\mathfrak{C}_1, Z_1)$. Define $l_{0,1} := \tilde{l}_{0,1} \pi_1 h_1 \sigma_1$. Then $l_{0,1}$ solves $(\Gamma_{5,1}, L)$.
Analogously define $\hat{l}_{0,2} : Z_2 \to X_2$ by permuting every $z \in \mathrm{Stab}^{\mathfrak{C}_2}(\pi_2 h_2 \sigma_2(V_{\bar{Z}}))$ with a different atom in X_2. It extends uniquely to a qf-automorphism $\tilde{l}_{0,2}$ of $(\mathfrak{C}_2, Z_2)$. Define $l_{0,2} := \tilde{l}_{0,2} \pi_2 h_2 \sigma_2$. Then $l_{0,2}$ solves $(\Gamma_{5,2}, L)$. This shows (1).

For every $x \in X_0$ by definition $\tilde{l}_{0,1}(x) = x$ and $\tilde{l}_{0,2}(x) = x$. Thus for variable $v \in V_Z$: $l_{0,2}(v) = \tilde{l}_{0,2} \pi_2 h_2 \sigma_2(v) = \pi_2 h_2 \sigma_2(v) = h_{1,2} \pi_1 h_1 \sigma_1(v) = h_{1,2} \tilde{l}_{0,1} \pi_1 h_1 \sigma_1(v) = h_{1,2} l_{0,1}(v)$. This shows (2).

$S_0 = \emptyset$, thus (3) is trivially true.

Finally $\mathrm{Stab}^{\mathfrak{C}_1}(l_{0,1}(V_{\bar{Z}})) \subset X_1$ and $\mathrm{Stab}^{\mathfrak{C}_2}(l_{0,2}(V_{\bar{Z}})) \subset X_2$ immediately by definition.

Induction Step: Let $Lab(v_{i+1}) = \Sigma_1$. – The argument for $Lab(v_{i+1}) = \Sigma_2$ is analogue. – We have to distinguish three cases.

Induction Step 1: $v_{i+1} \in V_Z$.
Define $l_{i+1,1} := l_{i,1}$ and $l_{i+1,2} := l_{i,2}$. Clearly, $l_{i+1,1}$ solves $(\Gamma_{5,1}, L)$ and $l_{i+1,2}$ solves $(\Gamma_{5,2}, L)$ by induction hypothesis. And for every variable $v \in V_Z$ it holds that $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v) \in X_0$ by induction hypothesis, too.

Because $l_{i+1,1}(v_{i+1}) \in X_0$, the shadow $\mathrm{Sd}(l_{i+1,1}(v_{i+1})) = \{l_{i+1,1}(v_{i+1})\}$. Hence $S_{i+1} = S_i \cup \{l_{i+1,1}(v_{i+1})\}$ by definition.
Let v be a variable such that $l_{i+1,1}(v) \in S_{i+1}$ or $l_{i+1,2}(v) \in S_{i+1}$.
Then $l_{i+1,1}(v) = l_{i,1}(v) \in S_i$ or $l_{i+1,2}(v) = l_{i,2}(v) \in S_i$ and thus $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis.
Or $l_{i+1,1}(v) = l_{i,1}(v) = l_{i,1}(v_{i+1})$ or $l_{i+1,2}(v) = l_{i,2}(v) = l_{i,2}(v_{i+1})$ and $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v) \in X_0$ was just shown.

For $k = 1, 2$ we have $l_{i+1,k} = l_{i,k}$ and therefore also $\mathrm{Stab}^{\mathfrak{C}_k}(l_{i+1,k}(V_{\bar{Z}})) = \mathrm{Stab}^{\mathfrak{C}_k}(l_{i,k}(V_{\bar{Z}}))$. Furthermore $S_i \cup X_k \subset S_{i+1} \cup X_k$. Thus (4) holds by induction hypothesis.

Induction Step 2: $v_{i+1} \in V_{\bar{Z}}$ and $l_{i,2}(v_{i+1}) = h_{1,2}l_{i,1}(v_{i+1})$.
This is another simple case. Again define $l_{i+1,1} := l_{i,1}$ and $l_{i+1,2} := l_{i,2}$. Properties (1) and (2) are satisfied by induction hypothesis.
Since $v_{i+1} \in V_{\bar{Z}}$ and $Lab(v_{i+1}) = \Sigma_1$, we know $l_{i,1}(v_{i+1})$ is a non-atom and $l_{i,2}(v_{i+1})$ is an atom. Therefore $\mathrm{Sd}(l_{i,2}(v_{i+1})) = \{l_{i,2}(v_{i+1})\} \cup \mathrm{Sd}(l_{i,1}(v_{i+1}))$ by definition of shadows and $S_{i+1} = S_i \cup \mathrm{Sd}(l_{i,2}(v_{i+1}))$. For a variable v such that $l_{i+1,1}(v) \in S_i$ or $l_{i+1,2}(v) \in S_i$ we have $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis. For the variable v_{i+1}, it is trivially the case. Remains the case that $l_{i,1}(v) \in \mathrm{Sd}(l_{i,1}(v_{i+1}))$ or $l_{i,2}(v) \in \mathrm{Sd}(l_{i,1}(v_{i+1}))$. Now, $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(v_{i+1})) \subset S_i \cup X_1$ by (4) of the induction hypothesis. Define $X_1' := X_1 \cap \mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(v_{i+1}))$. Then $\mathrm{Sd}(l_{i,1}(v_{i+1})) \subset S_i \cup X_1' \cup \{l_{i,1}(v_{i+1})\}$.
If $l_{i,1}(v) \in S_i$, then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis. The case $l_{i,1}(v) \in X_1'$ is impossible: Suppose $l_{i,1}(v) \in X_1'$, then $v <_L v_{i+1}$ by the linear constant restriction – $l_{i,1}(v) \in \mathrm{Stab}(l_{i,1}(v+1))$ is allowed only, if $v <_L v_{i+1}$. Then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v) \in X$ again by induction hypothesis. Therefore $h_1\sigma_1(v) \in Z_1$ and $h_2\sigma_2(v) \in Z_2$, because $l_{i,1}$ is a qf-isomorphism. Thus $v \in V_Z$ and $l_{i,1}(v) \in X_0$. Contradiction, since $X_0 \cap X_1 = \emptyset$.
If $l_{i,2}(v) \in S_i$, then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ again by induction hypothesis. The case $l_{i,2}(v) \in X_1'$ is impossible: Suppose $l_{i,2}(v) \in X_1'$, then $v <_L v_{i+1}$ by the linear constant restriction. Then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v) \in X$ again by induction hypothesis and we end in the same contradiction as above.

For $k = 1, 2$ we have $l_{i+1,k} = l_{i,k}$ and therefore also $\mathrm{Stab}^{\mathfrak{C}_k}(l_{i+1,k}(V_{\bar{Z}})) = \mathrm{Stab}^{\mathfrak{C}_k}(l_{i,k}(V_{\bar{Z}}))$. Furthermore $S_i \cup X_k \subset S_{i+1} \cup X_k$. Thus (4) holds by induction hypothesis.

Induction Step 3: $v_{i+1} \in V_{\bar{Z}}$ and $l_{i,2}(v_{i+1}) \neq h_{1,2}l_{i,1}(v_{i+1})$.
Let $y_{i+1} := l_{i,2}(v_{i+1}) \in Z_2$ and $z_{i+1} := h_{1,2}l_{i,1}(v_{i+1}) \in Z_2$. Define the following

permutation $t_{i+1} : Z_2 \to Z_2$ by $y_{i+1} \longleftrightarrow z_{i+1}$. Then t_{i+1} extends uniquely to a qf-automorphism τ_{i+1}. Define $l_{i+1,1} := l_{i,1}$ and $l_{i+1,2} := \tau_{i+1} l_{i,2}$. Then $l_{i+1,1}$ solves $(\Gamma_{5,1}, L)$ by induction hypothesis. And $l_{i+1,2}$ solves $(\Gamma_{5,2}, L)$ by Lemma 6.2.8 and induction hypothesis for $l_{i,2}$.

Furthermore, we can establish the following facts

(a) $z_{i+1} \in Z_2 \setminus X$ and especially $z_{i+1} \notin X_2$ or X_1.
This is a consequence of the definition of the fibring construction.

(b) $z_{i+1} \notin S_i$.
(If $z_{i+1} \in S_i$, then $h_{2,1}(z_{i+1}) = l_{i,1}(v_{i+1}) \in S_i$ by definition of shadows, then $z_{i+1} = h_{1,2} l_{i,1}(v_{i+1}) = l_{i,2}(v_{i+1}) = y_{i+1}$ by induction hypothesis.)

(c) $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V_{\bar{Z}}))$ and $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}}))$.
($\mathrm{Stab}^{\mathfrak{C}_k}(l_{i,k}(V_{\bar{Z}})) \subset S_i \cup X_k$, $k = 1, 2$.)

(d) $y_{i+1} \in X_2$, $y_{i+1} \notin S_i$.
($y_{i+1} \in S_i \cup X_2$ by induction hypothesis (4) and $y_{i+1} \in S_i \implies y_{i+1} = z_{i+1}$.)

By (a) and (d), τ_{i+1} is the identity on X_0. Therefore for all $v \in V_Z$: $l_{i+1,2}(v) = \tau l_{i,2}(v) = l_{i,2}(v) = h_{1,2} l_{i,1}(v) = h_{1,2} l_{i+1,1}(v) \in X_0$. This shows Property (2).

By (b) and (d), τ_{i+1} is the identity on S_i.
$\mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V)) = \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_Z)) \cup \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}}))$; by the above τ_{i+1} is the identity on $X_0 \supseteq \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_Z))$. By (c), $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}}))$; by (d), $y_{i+1} \in X_2$, hence $y_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}})) \subseteq S_i \cup X_1$, because $X_1 \cap X_2 = \emptyset$. To sum up, τ_{i+1} is the identity on $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V))$ and therefore on $l_{i,1}(V)$.

By definition, $S_{i+1} = \bigcup_{j \leq i} \mathrm{Sd}(l_{i+1,1}(v_j)) \cup \bigcup_{j \leq i} \mathrm{Sd}(l_{i+1,2}(v_j)) \cup \mathrm{Sd}(l_{i+1,1}(v_{i+1})) \cup \mathrm{Sd}(l_{i+1,2}(v_{i+1}))$. Now, τ_{i+1} is the identity on S_i, thus $\bigcup_{j \leq i} \mathrm{Sd}(l_{i+1,1}(v_j)) \cup \bigcup_{j \leq i} \mathrm{Sd}(l_{i+1,2}(v_j)) = S_i$. And $\mathrm{Sd}(l_{i+1,2}(v_{i+1})) = \{z_{i+1}\} \cup \mathrm{Sd}(l_{i+1,1}(v_{i+1}))$ by definition. Hence $S_{i+1} = S_i \cup \mathrm{Sd}(z_{i+1})$.
For a variable v, if $l_{i+1,1}(v) = l_{i,1}(v) \in S_i$ then $l_{i+1,2}(v) = l_{i,2}(v) = h_{1,2} l_{i+1,1}(v)$ by induction hypothesis.
If $l_{i+1,2}(v) \in S_i$, then $l_{i+1,2}(v) = l_{i,2}(v)$. Therefore $l_{i+1,2}(v) = h_{1,2} l_{i+1,1}(v)$ by induction hypothesis.

As seen, $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(V))$, and therefore also $z_{i+1} \notin l_{i+1,1}(V)$; hence $l_{i+1,1}(v) \neq z_{i+1}$. Thus, if $l_{i+1,1}(v) \in \mathrm{Sd}(z_{i+1})$, then $l_{i+1,1}(v) \in \mathrm{Sd}(l_{i+1,1}(v_{i+1}))$. Now, $\mathrm{Stab}(l_{i+1,1}(v_{i+1})) = \mathrm{Stab}(l_{i,1}(v_{i+1}) \subset S_i \cup X_1$ by induction hypothesis (4). Define $X_1' := X_1 \cap \mathrm{Stab}(l_{i+1,1}(v_{i+1}))$. Then $\mathrm{Sd}(l_{i+1,1}(v_{i+1})) \subset \{l_{i+1,1}(v_{i+1})\} \cup X_1' \cup S_i$. The case $l_{i+1,1}(v) = l_{i+1,1}(v_{i+1})$ is simple, because as a consequence of Lemma 6.2.7 $v = v_{i+1}$. The case $l_{i+1,1}(v) \in S_i$ is already taken care of. The case $l_{i+1,1}(v) \in X_1'$ is impossible: Suppose $l_{i+1,1}(v) \in X_1'$, then $v <_L v_{i+1}$ by the linear constant restriction. Thus $l_{i+1,2}(v) = h_{1,2} l_{i+1,1}(v) \in X$. Thus both $h_1 \sigma_1(v) \in Z_1$ and $h_2 \sigma_2(v) \in Z_2$. Therefore $v \in V_Z$ and $l_{i+1,1}(v) \in X_0$ which is a contradiction since $X_0 \cap X_1 = \emptyset$.

If $l_{i+1,2}(v) \in \mathrm{Sd}(l_{i+1,2}(v_{i+1}))$ then $l_{i+1,2}(v) = z_{i+1}$ or $l_{i+1,2}(v) \in S_i$. $l_{i+1,2}(v) = l_{i+1,1}(v_{i+1})$ is impossible, because $l_{i+1,1}(v_{i+1}) \in C_1 \setminus Z_1$ and $l_{i+1,2}(v) \in C_2$. And

$l_{i+1,2}(v) \notin X_1'$, because $l_{i,2}(v) \notin X_1$ by induction hypothesis (4) and $z_{i+1} \notin X_1$ by (a). The case $l_{i+1,2}(v) = z_{i+1}$ is simple, because τ_{i+1} is defined such that $z_{i+1} = h_{1,2} l_{i+1,1}(v_{i+1})$. The case of $l_{i+1,2}(v) \in S_i$ was handled above.

Finally, since $l_{i+1,1} = l_{i,1}$ we have $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(V_{\bar{Z}})) = \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V_{\bar{Z}})) \subset S_i \cup X_1 \subset S_{i+1} \cup X_1$. And $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(V_{\bar{Z}})) \subseteq (\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V_{\bar{Z}})) \setminus \{y_{i+1}\}) \cup \{z_{i+1}\}$ by Lemma 6.2.6. Since $z_{i+1} \in S_{i+1}$, we have $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(V_{\bar{Z}})) \subset S_{i+1} \cup X_2$. And this ends the induction proof.

By this induction, we showed that for $i = n$ there are assignments $l_{n,1}$ and $l_{n,2}$ such that $l_{n,1}$ solves $(\Gamma_{5,1}, L)$ and $l_{n,2}$ solves $(\Gamma_{5,2}, L)$ and for every variable $v \in V : l_{n,2}(v) = h_{1,2} l_{n,1}(v)$. Therefore $l_{n,1}$ solves $\Gamma_{5,1}$ and $\Gamma_{5,2}$ in $\mathfrak{C}_1^{\Sigma}$ and hence also Γ_4. Thus there is a solution of Γ_0 in $\mathfrak{C}_1^{\Sigma}$ by assigning each variable that value that its representant receives by $l_{n,1}$. ■

Definition 6.2.11 As in Proposition 6.2.5 above, let σ_1 be a solution of $(\Gamma_{5,1}, L)$ and σ_2 a solution of $(\Gamma_{5,2}, L)$. Define $\sigma_1 \otimes \sigma_2$ as the solution of Γ_0 as constructed in the proof above.

Corollary 6.2.12 *There exists a Σ_1-isomorphism that maps σ_1 to $\sigma_1 \otimes \sigma_2$ and there exists a Σ_2-isomorphism that maps σ_2 to $\sigma_1 \otimes \sigma_2$.*

Completeness

Proposition 6.2.13 *If the input problem Γ_0 is solvable, then there exists a linear constant restriction $L = (Lab, <)$ and an output pair $(\Gamma_{5,1}, \Gamma_{5,2})$ such that both $(\Gamma_{5,1}, L)$ and $(\Gamma_{5,2}, L)$ are solvable.*

Proof. Let σ be the solution in the combined solution domain $\mathfrak{C}_1^{\Sigma}$ with atom set Z_1. In Step 3, we identify two variables x and y, iff $\sigma(x) = \sigma(y)$. In Step 4, variable x receives label Σ_2, iff $\sigma(x) \in Z_1$. Otherwise x receives label Σ_1. And the order $<$ is defined by: $x < y$, iff $\sigma(x) <_i \sigma(y)$ according to Definition 3.3.9 (on page 35). This gives the linear constant restriction L. Then σ obviously solves Γ_4 and therefore $\Gamma_{5,1}$ and $\Gamma_{5,2}$.

Let $h_1 : \mathfrak{C}_1^{\Sigma_1} \to \mathfrak{A}_1^{\Sigma_1}$ be the qf-isomorphism between $\mathfrak{C}_1^{\Sigma_1}$ and $\mathfrak{A}_1^{\Sigma_1}$. Then $h_1 \circ \sigma$ solves $\Gamma_{5,1}$ in $\mathfrak{A}_1^{\Sigma_1}$.
For the linear constant restriction L:
If $Lab(x) = \Sigma_2$, then $\sigma(x) \in Z_1$ by definition of L and $h_1 \circ \sigma(x) \in X$ by definition of a qf-isomorphism.
If $Lab(x) = \Sigma_2$ and $Lab(y) = \Sigma_2$ and $x \neq y$, then $\sigma(x) \neq \sigma(y)$ by definition of the identification in Step 3 and $h_1 \circ \sigma(x) \neq h_1 \circ \sigma(y)$, because h_1 is an isomorphism.
If $x < y$, $Lab(x) = \Sigma_1$ and $Lab(y) = \Sigma_2$, then by definition of the order $<$, $\sigma(x)$ is fibred before $\sigma(y)$ and thus $\sigma(y) \notin \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(\sigma(x))$. Therefore $h_1 \circ \sigma(y) \notin \mathrm{Stab}_{\Sigma_1}^{\mathfrak{A}_1}(h_1 \circ \sigma(x))$.

Let $h_2 : \mathfrak{C}_2^{\Sigma_2} \to \mathfrak{A}_2^{\Sigma_2}$ be the qf-isomorphism between $\mathfrak{C}_2^{\Sigma_2}$ and $\mathfrak{A}_2^{\Sigma_2}$. Then $h_2 \circ h_{1,2} \circ \sigma$ solves $\Gamma_{5,2}$ in $\mathfrak{A}_2^{\Sigma_2}$.
For the linear constant restriction L:
If $Lab(x) = \Sigma_1$, then $\sigma(x) \in C_1 \setminus Z_1$, therefore $h_{1,2} \circ \sigma(x) \in Z_2$ by definition of the fibring construction and $h_2 \circ h_{1,2} \circ \sigma(x) \in X$, since h_2 is a qf-isomorphism.
If $Lab(x) = \Sigma_1$ and $Lab(y) = \Sigma_1$ and $x \neq y$, then $\sigma(x) \neq \sigma(y)$ by definition of the identification in Step 3. Thus $h_2 \circ h_{1,2} \circ \sigma(x) \neq h_2 \circ h_{1,2} \circ \sigma(y)$, because $h_2 \circ h_{1,2}$ is an isomorphism.
If $x < y$, $Lab(x) = \Sigma_2$ and $Lab(y) = \Sigma_1$, then by definition of the order $<$, $\sigma(x)$ is fibred before $\sigma(y)$ and thus $h_{1,2} \circ \sigma(y) \notin \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(h_{1,2} \circ \sigma(x))$. Therefore $h_2 \circ h_{1,2} \circ \sigma(y) \notin \mathrm{Stab}_{\Sigma_2}^{\mathfrak{A}_2}(h_2 \circ h_{1,2} \circ \sigma(x))$. ■

6.3 Free Amalgamation of Negative Constraints: Ground Solvability

To discuss ground solvability, we first have to extend the notion of a ground solution known from equational unification to the more general case of quasi-free structures. In case, where we only have a constraint problem Γ without linear constant restriction, the notion of a ground solution of Γ is a straight forward generalisation of the definition for equational theories. In an equational theory, a solution is ground, if no solution term contains a variable. For quasi-free structures, this just means, that the stabiliser of every solution element has to be empty. In case, where we consider constraint problems (Γ, L) with linear constant restriction L, things are a bit more complicated. In equational theories, all such variables that are assigned alien labels are considered as constants, thus even if they appear in a solution term that contains no other variables, this term is still considered as ground. Analogously for quasi-free structures, we will permit the appearance of such atoms in the stabiliser of a solution element that are themselves different solution elements. If we demanded here, too, that stabilisers have to be empty, we would yield only trivial combination problems with no really mixed elements. We also define the notion of a restrictive solution for quasi-free structures. This notion was introduced by F. Baader & K. U. Schulz in [9] for equational disunification problems. There it said that a solution is restrictive, if whenever a variable is assigned a complex term in the solution, then this term is not equivalent (modulo the equational theory) to a variable.

Definition 6.3.1 Let Γ be a constraint problem. A solution is called *ground*, iff every solution element has an empty stabiliser.
Let (Γ, L) be a constraint problem with linear constant restriction L over signature Σ. A solution σ is called a *ground* solution, iff for every variable $v \in Var(\Gamma)$ such that $Lab(v) = \Sigma$: $\sigma(v)$ is not an atom and $\mathrm{Stab}(\sigma(v)) \subseteq \sigma(Var(\Gamma) \setminus \{v\})$. That is the stabiliser of a non-atom solution element can only consist of atoms that are themselves solutions of variables different from v.

A solution σ is called *restrictive*, iff for every variable $v \in Var(\Gamma)$ such that $Lab(v) = \Sigma$: $\sigma(v)$ is not an atom.

For a ground solution, note that due to the linear constant restriction, the only atoms in the stabiliser of a non-atom solution element allowed stem from solutions of variables that are smaller according to the given linear order. Obviously, every ground solution is restrictive.

This section's main statement is the following

Theorem 6.3.2 *Let $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$ be two quasi-free structures over disjoint signatures with infinitely many ground elements. The existential theory of the ground substructure of the free amalgamated product $\mathfrak{A}_1^{\Sigma_1} \otimes \mathfrak{A}_2^{\Sigma_2}$ is decidable, if the existence of restrictive solutions of pure constraint problems with linear constant restrictions is decidable in both components $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$.*

Remember that a ground substructure (Def. 3.2.18 on page 25) of a quasi-free structure is just the structure of its ground elements. It would of course be desirable to reduce ground solvability of mixed constraints in the free amalgam to ground solvability in the components. But the decomposition algorithm given in the last section does not permit so. We can proof a soundness proposition:

Proposition 6.3.3 *Let Γ_0 be the input problem of the Decomposition Algorithm 6.2.4. Suppose there exists an output pair $(\Gamma_{5,1}, \Gamma_{5,2})$ and a linear constant restriction L such that both $(\Gamma_{5,1}, L)$ and $(\Gamma_{5,2}, L)$ have ground solutions. Then Γ_0 has a ground solution.*

Proof. Let $\mathfrak{C}_1^{\Sigma}$ be the combined solution domain of $\mathfrak{A}_1^{\Sigma_1}$ and $A_2^{\Sigma_2}$, and let σ_1 be the ground solution of $(\Gamma_{5,1}, L)$ in $\mathfrak{A}_1^{\Sigma_1}$ and σ_2 the ground solution of $(\Gamma_{5,2}, L)$ in $\mathfrak{A}_2^{\Sigma_2}$ where $L = (Lab, <)$. Let $h_{1,2} : \mathfrak{C}_1^{\Sigma} \to \mathfrak{C}_2^{\Sigma}$ be the qf-isomorphism between $\mathfrak{C}_1^{\Sigma}$ and $\mathfrak{C}_2^{\Sigma}$.

Let $h_1 : \mathfrak{A}_1^{\Sigma_1} \to \mathfrak{C}_1^{\Sigma_1}$ be the qf-isomorphism between $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{C}_1^{\Sigma_1}$. Then $h_1\sigma_1$ is a ground solution of $(\Gamma_{5,1}, L)$ in $\mathfrak{C}_1^{\Sigma_1}$ ($\mathfrak{C}_1^{\Sigma}$) by Lemma 6.2.8.

Let $h_2 : \mathfrak{A}_2^{\Sigma_2} \to \mathfrak{C}_2^{\Sigma_2}$ be the qf-isomorphism between $\mathfrak{A}_2^{\Sigma_2}$ and $\mathfrak{C}_2^{\Sigma_2}$. Then $h_2\sigma_2$ is a ground solution of $(\Gamma_{5,2}, L)$ in $\mathfrak{C}_2^{\Sigma_2}$ by Lemma 6.2.8.

Let $v_1, \ldots, v_n$ be the enumeration of the variables V along the ordering $<$, i. e. $v_i < v_j$, iff $i < j$. Note that there is no variable $v \in V$ with $h_1\sigma_1(v) \in Z_1$ and $h_2\sigma_2(v) \in Z_2$, because if $Lab(v) = \Sigma_i$ (for $i = 1$ or $i = 2$), then $h_i\sigma_i(v)$ is a non-atom, since $h_i\sigma_i$ is a ground solution.

We will now prove by induction over the variables $v_1, v_2, \ldots, v_n$ as given by the linear order $<$ that for $i = 0, \ldots, n$ the following 3 properties hold.

1. There is an assignment $l_{i,1} : V \to C_1$ such that $l_{i,1}$ is a ground solution of $(\Gamma_{5,1}, L)$ and there is an assignment $l_{i,2} : V \to C_2$ such that $l_{i,2}$ is a ground solution of $(\Gamma_{5,2}, L)$.

2. For every $j \leq i$: $l_{i,2}(v_j) = h_{1,2}l_{i,1}(v_j)$.

3. For every $j \leq i$: $l_{i,1}(v_j)$ is ground in $(\mathfrak{C}_1^\Sigma, X)$.

Induction Base:
Define $l_{0,1} := h_1\sigma_1$ and $l_{0,2} := h_2\sigma_2$. Then (1) holds by the remarks above and (2) and (3) are trivially true.

Induction Step: Let $Lab(v_{i+1}) = \Sigma_1$. – The argument for $Lab(v_{i+1}) = \Sigma_2$ is analogue.
Define the following map: $\hat{l}_{i+1} : Z_2 \to Z_2$ by the transposition $l_{i,2}(v_{i+1}) \longleftrightarrow h_{1,2}l_{i,1}(v_{i+1})$. This permutation extends uniquely to a qf-automorphism $\tilde{l}_{i+1}$ of $\mathfrak{C}_2^{\Sigma_2}$. Define $l_{i+1,2} := \tilde{l}_{i+1}l_{i,2}$ and $l_{i+1,1} := l_{i,1}$. Then Property (1) holds by Lemma 6.2.8. And obviously by definition of $\tilde{l}_{i+1}$ we have $l_{i+1,2}(v_{i+1}) = h_{1,2}l_{i+1,1}(v_{i+1})$.

Now $l_{i,2}(v_{i+1}) \notin \mathrm{Stab}^{\mathfrak{C}_2}(\{l_{i,2}(v_1), \ldots, l_{i,2}(v_i)\})$: If $v_j < v_{i+1}$ and $Lab(v_j) = \Sigma_1$ then $l_{i,2}(v_j) \in Z_2$ and $l_{i,2}(v_j) \neq l_{i,2}(v_{i+1})$ because $l_{i,2}$ is the composition of injective functions. If $Lab(v_j) = \Sigma_2$ then $l_{i,2}(v_{i+1}) \notin \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v_j))$ due to the linear constant restriction.
And $h_{1,2}l_{i,1}(v_{i+1}) \notin \mathrm{Stab}^{\mathfrak{C}_2}(\{l_{i,2}(v_1), \ldots, l_{i,2}(v_i)\})$: If $v_j < v_{i+1}$ and $Lab(v_j) = \Sigma_1$ then $l_{i,2}(v_j) \in Z_2$ and $l_{i,2}(v_j) = h_{1,2}l_{i,1}(v_j)$ by induction hypothesis. Now since $v_j \neq v_{i+1}$ we have $h_{1,2}l_{i,1}(v_j) \neq h_{1,2}l_{i,1}(v_{i+1})$, because $h_{1,2}l_{i,1}$ is the composition of injective functions. Thus $l_{i,2}(v_j) \neq h_{1,2}l_{i,1}(v_{i+1})$. If $Lab(v_j) = \Sigma_2$ then for every $z \in \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v_j))$ exists a variable $v_k < v_j$ with $z = l_{i,2}(v_k)$ by definition of a ground-solution. Therefore $h_{1,2}l_{i,1}(v_{i+1}) \notin \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v_j))$ by the same argument as above.
Hence $\tilde{l}_{i+1}$ is the identity on $l_{i,2}(v_1), \ldots, l_{i,2}(v_i)$. Thus for $v_j < v_{i+1}$ we have $l_{i+1,2}(v_j) = l_{i,2}(v_j) = h_{1,2}l_{i,1}(v_j) = h_{1,2}l_{i+1,1}(v_j)$ by induction hypothesis. This shows Property (2).

Because for $j < i+1$ we know $\tilde{l}_{i+1}$ is the identity on $l_{i,2}(v_j)$, we know $l_{i+1,1}(v_j)$ is ground by induction hypothesis. Because for every $z \in \mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(v_{i+1}))$ there is a $v < v_{i+1}$ with $z = l_{i+1,1}(v)$ and this $l_{i+1,1}(v)$ is ground by the above, $l_{i+1,1}(v_{i+1})$ is ground, too. And this proof of Property (3) completes the induction.

For $i = n$ the above induction shows that $l_{n,1}$ is a ground solution of $(\Gamma_{5,1}, L)$ and $l_{n,2}$ is a ground solution of $(\Gamma_{5,2}, L)$. Furthermore for all variables v: $l_{n,2}(v) = h_{1,2}l_{n,1}(v)$. Thus $l_{n,1}$ is a ground solution of $\Gamma_{5,2}$ in $\mathfrak{C}_1^\Sigma$. And for each variable v, the element $l_{n,1}(v)$ is ground in $(\mathfrak{C}_1^\Sigma, X)$. ■

This shows soundness of the decomposition algorithm. Unfortunately, this algorithm is not complete. In [9], Baader and Schulz give an example for equational disunification (4.2 on page 243) that demonstrates this fact. Therefore we have to demand the existence of restricted solutions in the components in order to get ground solvability in the combined domain.

Proposition 6.3.4 *Assume that both $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$ contain an infinite number of ground elements*[2]*. Let Γ_0 be the input problem of the decomposition algorithm. Suppose there exists an output pair $(\Gamma_{5,1}, \Gamma_{5,2})$ and a linear constant restriction L such that both $(\Gamma_{5,1}, L)$ and $(\Gamma_{5,2}, L)$ have restricted solutions. Then Γ_0 has a ground solution.*

Before we can present the proof, we give a little technical lemma that we will make use of in the proof.

Lemma 6.3.5 *Let $(\mathfrak{C}_1^{\Sigma}, X)$ be the free amalgamated product of $(\mathfrak{A}_1^{\Sigma_1}, X)$ and $(\mathfrak{A}_2^{\Sigma_2}, X)$. An element $c \in C_1$ is ground in $(\mathfrak{C}_1^{\Sigma}, X)$, iff $\mathrm{Bottom}(\mathrm{Sd}(c)) \cap X = \emptyset$, i.e., there are no atoms amongst the bottom elements of its shadow.*

Proof. We prove this lemma by an induction over the depth of the shadow of c, where the depth of the shadow is just the length of the longest branch in the shadow.
In the base case, c is a bottom element itself. Thus $\mathrm{Bottom}(\mathrm{Sd}(c)) = \{c\}$ and c is ground in $(\mathfrak{C}_1^{\Sigma}, X)$ iff it is ground in $(\mathfrak{C}_1^{\Sigma_1}, Z_1)$.
In the step case, c is not a bottom element.
If $c \notin Z_1$ is a non-atom and ground in $(\mathfrak{C}_1^{\Sigma}, X)$, then each element in $\mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(c)$ is also ground in $(\mathfrak{C}_1^{\Sigma}, X)$. By induction hypothesis, for each $z \in \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(c)$: $\mathrm{Bottom}(\mathrm{Sd}(z)) \cap X = \emptyset$. Since $\mathrm{Sd}(c) = \{c\} \cup \{\mathrm{Sd}(z) \mid z \in \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(c)$ we have also $\mathrm{Bottom}(\mathrm{Sd}(c)) \cap X = \emptyset$.
If $c \in Z_1$ is an atom and ground in $(\mathfrak{C}_1^{\Sigma}, X)$, then $h_{1,2}(c)$ is ground in $(\mathfrak{C}_2^{\Sigma}, X)$, therefore each element in $\mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(h_{1,2}(c))$ is also ground in $(\mathfrak{C}_2^{\Sigma}, X)$. By induction hypothesis, for each $z \in \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(h_{1,2}(c))$: $\mathrm{Bottom}(\mathrm{Sd}(z)) \cap X = \emptyset$. Hence $\mathrm{Bottom}(\mathrm{Sd}(h_{1,2}(c))) \cap X = \emptyset$ and $\mathrm{Bottom}(\mathrm{Sd}(c)) \cap X = \emptyset$.
If $c \notin Z_1$ is a non-atom and $\mathrm{Bottom}(\mathrm{Sd}(c)) \cap X = \emptyset$, then for each $z \in \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(c)$: $\mathrm{Bottom}(\mathrm{Sd}(z)) \cap X = \emptyset$. By induction hypothesis, each $z \in \mathrm{Stab}_{\Sigma_1}^{\mathfrak{C}_1}(c)$ is ground in $(\mathfrak{C}_1^{\Sigma}, X)$. Therefore c is ground in $(\mathfrak{C}_1^{\Sigma}, X)$.
If $c \in Z_1$ is an atom and $\mathrm{Bottom}(\mathrm{Sd}(c)) \cap X = \emptyset$, then $\mathrm{Bottom}(\mathrm{Sd}(h_{1,2}(c))) \cap X = \emptyset$ and for each $z \in \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(h_{1,2}(c))$: $\mathrm{Bottom}(\mathrm{Sd}(z)) \cap X = \emptyset$. By induction hypothesis, each $z \in \mathrm{Stab}_{\Sigma_2}^{\mathfrak{C}_2}(h_{1,2}(c))$ is ground in $(\mathfrak{C}_2^{\Sigma}, X)$. Therefore $h_{1,2}(c)$ is ground in $(\mathfrak{C}_2^{\Sigma}, X)$ and c is ground in $(\mathfrak{C}_1^{\Sigma}, X)$. ■

The following proof is very similar to the soundness proof in the non-ground case. The first difference is that due to the fact that the component solutions are restricted, there is no variable which is assigned to an atom in both component solutions. Hence no special treatment of those variables is required. Then, we have to show that we always work with restricted solutions, but that is simple, because a qf-isomorphism obviously maps atoms to atoms and non-atoms to non-atoms and thus preserves the property of a solution being restrictive. Finally, we must show that the combined solution that we receive is actually ground in $(\mathfrak{C}_1^{\Sigma}, X)$.

[2]In the following, a ground element is always an element with empty stabiliser.

Proof of Proposition 6.3.4.
Let $\mathfrak{C}_1^{\Sigma}$ be the free amalgamated product of $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{A}_2^{\Sigma_2}$, and let σ_1 be the restrictive solution of $(\Gamma_{5,1}, L)$ in $\mathfrak{A}_1^{\Sigma_1}$ and σ_2 the restrictive solution of $(\Gamma_{5,2}, L)$ in $\mathfrak{A}_2^{\Sigma_2}$ where $L = (Lab, <_L)$. Let $h_{1,2} : \mathfrak{C}_1^{\Sigma} \to \mathfrak{C}_2^{\Sigma}$ be the qf-isomorphism between $\mathfrak{C}_1^{\Sigma}$ and $\mathfrak{C}_2^{\Sigma}$.

Let $h_1 : \mathfrak{A}_1^{\Sigma_1} \to \mathfrak{C}_1^{\Sigma_1}$ be the qf-isomorphism between $\mathfrak{A}_1^{\Sigma_1}$ and $\mathfrak{C}_1^{\Sigma_1}$. Then $h_1\sigma_1$ is a restrictive solution of $(\Gamma_{5,1}, L)$ in $\mathfrak{C}_1^{\Sigma_1}$ ($\mathfrak{C}_1^{\Sigma}$) by Lemma 6.2.8.

Let $h_2 : \mathfrak{A}_2^{\Sigma_2} \to \mathfrak{C}_2^{\Sigma_2}$ be the qf-isomorphism between $\mathfrak{A}_2^{\Sigma_2}$ and $\mathfrak{C}_2^{\Sigma_2}$. Then $h_2\sigma_2$ is a restrictive solution of $(\Gamma_{5,2}, L)$ in $\mathfrak{C}_2^{\Sigma_2}$ by Lemma 6.2.8.

Let $v_1, \dots, v_n$ be the enumeration of the variables V along the ordering $<_L$, i. e. $v_i <_L v_j$, iff $i < j$. Note that there is no variable $v \in V$ with $h_1\sigma_1(v) \in Z_1$ and $h_2\sigma_2(v) \in Z_2$, because if $Lab(v) = \Sigma_i$ (for $i = 1$ or $i = 2$), then $h_i\sigma_i(v)$ is a non-atom, since $h_i\sigma_i$ is a restrictive solution.

Let $G_1 \subset C_1$ be the set of all ground elements with respect to $(\mathfrak{C}_1^{\Sigma_1}, Z_1)$ such that they are not solutions of $h_1\sigma_1$. This set is infinite, since $\mathfrak{A}_1$ has infinitely many ground elements. Define $Y_2 := \{y \in Z_2 \mid \exists b \in G_1 : y = h_{1,2}(b)\} \setminus \mathrm{Stab}^{\mathfrak{C}_2}(h_2\sigma_2(V))$. Y_2 again is infinite. Analogously, let $G_2 \subset C_2$ be the set of all ground elements with respect to $(\mathfrak{C}_2^{\Sigma_2}, Z_2)$ such that they are not solutions of $h_2\sigma_2$. This set is infinite, since $\mathfrak{A}_2$ has infinitely many ground elements. Define $Y_1 := \{y \in Z_1 \mid \exists b \in G_2 : y = h_{2,1}(b)\} \setminus \mathrm{Stab}^{\mathfrak{C}_1}(h_1\sigma_1(V))$. Y_1 again is infinite.

We will now prove by induction over the variables $v_1, v_2, \dots, v_n$ as given by the linear order $<_L$ that for $i = 0, \dots, n$ the following 4 properties hold.

1. There is an assignment $l_{i,1} : V \to C_1$ such that $l_{i,1}$ is a restrictive solution of $(\Gamma_{5,1}, L)$ and there is an assignment $l_{i,2} : V \to C_2$ such that $l_{i,2}$ is a restrictive solution of $(\Gamma_{5,2}, L)$.

2. Define
$$S_i := \bigcup_{j \le i} \mathrm{Sd}(l_{i,1}(v_j)) \cup \bigcup_{j \le i} \mathrm{Sd}(l_{i,2}(v_j)).$$
For every variable v such that $l_{i,1}(v) \in S_i$ or $l_{i,2}(v) \in S_i$ holds $l_{i,2}(v) = h_{1,2}l_{i,1}(v)$.

3. $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V)) \subseteq S_i \cup Y_1$, $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V)) \subseteq S_i \cup Y_2$.

4. For all $v \in V$: If $Lab(v) = \Sigma_1$ then $\mathrm{Bottom}(\mathrm{Sd}(l_{i,1}(v))) \cap X = \emptyset$, if $Lab(v) = \Sigma_2$ then $\mathrm{Bottom}(\mathrm{Sd}(l_{i,2}(v))) \cap X = \emptyset$.

Induction Base:
Define firstly the following map $\hat{l}_{0,1} : Z_1 \to Y_1$ by permuting every $z \in \mathrm{Stab}^{\mathfrak{C}_1}(h_1\sigma_1(V))$ with a different atom in Y_1. It extends uniquely to a qf-automorphism $\tilde{l}_{0,1}$ of $(\mathfrak{C}_1, Z_1)$. Define $l_{0,1} := \tilde{l}_{0,1}h_1\sigma_1$. Then $l_{0,1}$ is a restrictive solution of $(\Gamma_{5,1}, L)$.
Analogously define the following map $\hat{l}_{0,2} : Z_2 \to Y_2$ by permuting every $z \in \mathrm{Stab}^{\mathfrak{C}_2}(h_2\sigma_2(V))$ with a different atom in Y_2. It extends uniquely to a

qf-automorphism $\tilde{l}_{0,2}$ of $(\mathfrak{C}_2, Z_2)$. Define $l_{0,2} := \tilde{l}_{0,2}h_2\sigma_2$. Then $l_{0,2}$ is a restrictive solution of $(\Gamma_{5,2}, L)$. This shows Property (1).

$S_0 = \emptyset$, thus Property (2) is quite trivially true. And $\mathrm{Stab}^{\mathfrak{C}_1}(l_{0,1}(V)) \subset Y_1$ and $\mathrm{Stab}^{\mathfrak{C}_2}(l_{0,2}(V)) \subset Y_2$ by definition.

Property (4) follows immediately from the above because Y_1 and Y_2 are defined as fibre images of ground elements in G_2 resp. G_1.

Induction Step: Let $Lab(v_{i+1}) = \Sigma_1$. – The argument for $Lab(v_{i+1}) = \Sigma_2$ is analogue. – We have to distinguish two cases.

Induction Step 1: $l_{i,2}(v_{i+1}) = h_{1,2}l_{i,1}(v_{i+1})$.
This is a simple case. Just define $l_{i+1,1} := l_{i,1}$ and $l_{i+1,2} := l_{i,2}$. Property (1) is satisfied by the induction hypothesis. $l_{i,2}(v_{i+1}) \in Z_2$, hence $l_{i,1}(v_{i+1}) \in \mathrm{Sd}(l_{i+1,2}(v_{i+1}))$. Therefore $S_{i+1} = S_i \cup \mathrm{Sd}(l_{i,2}(v_{i+1}))$. For a variable v such that $l_{i+1,1}(v) \in S_i$ or $l_{i+1,2}(v) \in S_i$ we have $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis. For the variable v_{i+1}, it is trivially the case. Now, $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(v_{i+1})) \subset S_i \cup Y_1$ by Property (3) of the induction hypothesis. If $l_{i+1,1}(v)$ or $l_{i+1,2}(v) \in S_i$, then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis. $l_{i+1,2}(v) \notin G_2$: Suppose it were, then $l_{i+1,2}(v) = h_2\sigma_2(v)$, since $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(v)) = \emptyset$. But that contradicts the choice of G_2 (not containing any solution elements). $l_{i+1,1}(v) \notin Y_1$: Suppose it were, then $v < v_{i+1}$ due to the linear constant restriction. Then $l_{i+1,2}(v) = h_{1,2}l_{i,1}(v)$ by induction hypothesis. But then $l_{i+1,2}(v) \in G_2$ by definition of Y_1, and we get the same contradiction as above. This shows Property (2). Properties (3) and (4) are true by induction hypothesis and $l_{i+1,k} = l_{i,k}$ $(k = 1, 2)$.

Induction Step 2: $l_{i,2}(v_{i+1}) \neq h_{1,2}l_{i,1}(v_{i+1})$.
Let $y_{i+1} := l_{i,2}(v_{i+1}) \in Z_2$ and $z_{i+1} := h_{1,2}l_{i,1}(v_{i+1}) \in Z_2$. Define the following permutation $t_{i+1} : Z_2 \to Z_2$ by $y_{i+1} \longleftrightarrow z_{i+1}$. Then t_{i+1} extends uniquely to a qf-automorphism τ_{i+1}. Define $l_{i+1,1} := l_{i,1}$ and $l_{i+1,2} := \tau_{i+1}l_{i,2}$. Then $l_{i+1,1}$ is a restrictive solution of $(\Gamma_{5,1}, L)$ by induction hypothesis. And $l_{i+1,2}$ is a restrictive solution of $(\Gamma_{5,2}, L)$ by Lemma 6.2.8 and induction hypothesis for $l_{i,2}$.

Furthermore, we can establish the following facts

(a) $z_{i+1} \notin Y_2$, $z_{i+1} \notin Y_1$.
(If $z_{i+1} \in Y_2$, then $l_{i,1}(v_{i+1}) \in G_1$, then $h_1\sigma_1(v_{i+1}) \in G_1$, which contradicts the choice of G_1. z_{i+1} is the fibre image of the non-atom $l_{i,1}(v_{i+1})$, hence $z_{i+1} \in Z_2 \setminus X$ and $(Z_2 \setminus X) \cap Z_1 = \emptyset$.)

(b) $z_{i+1} \notin S_i$.
(If $z_{i+1} \in S_i$, then $y_{i+1} = z_{i+1}$ by induction hypothesis.)

(c) $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V))$ and $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V))$.
($\mathrm{Stab}^{\mathfrak{C}_k}(l_{i,k}(V)) \subset S_i \cup Y_k$, $k = 1, 2$.)

(d) $y_{i+1} \in Y_2$, $y_{i+1} \notin S_i$, $y_{i+1} \notin X$, $y_{i+1} \notin Y_1$.
($y_{i+1} \in S_i \cup Y_2$ by induction hypothesis (4) and $y_{i+1} \in S_i \Longrightarrow y_{i+1} = z_{i+1}$.)

By (c) and (d), τ_{i+1} is the identity on $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V))$. By (b) and (d), τ_{i+1} is the identity on S_i.

By definition, $S_{i+1} = \bigcup_{j\leq i} \mathrm{Sd}(l_{i+1,1}(v_j)) \cup \bigcup_{j\leq i} \mathrm{Sd}(l_{i+1,2}(v_j)) \cup \mathrm{Sd}(l_{i+1,1}(v_{i+1})) \cup \mathrm{Sd}(l_{i+1,2}(v_{i+1}))$. Now, τ_{i+1} is the identity on S_i, thus $\bigcup_{j\leq i} \mathrm{Sd}(l_{i+1,1}(v_j)) \cup \bigcup_{j\leq i} \mathrm{Sd}(l_{i+1,2}(v_j)) = S_i$. And $\mathrm{Sd}(l_{i+1,2}(v_{i+1})) = \{z_{i+1}\} \cup \mathrm{Sd}(l_{i+1,1}(v_{i+1}))$ by definition. Hence $S_{i+1} = S_i \cup \mathrm{Sd}(z_{i+1})$.
For a variable v, if $l_{i+1,1}(v) = l_{i,1}(v) \in S_i$ then $l_{i+1,2}(v) = l_{i,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis.
If $l_{i+1,2}(v) \in S_i$, then $l_{i+1,2}(v) = l_{i,2}(v)$. Therefore $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis.
If $l_{i+1,1}(v) \in \mathrm{Sd}(z_{i+1})$ then $l_{i+1,1}(v) = l_{i+1,1}(v_{i+1})$ or $l_{i+1,1}(v) \in S_i$, because $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(v_{i+1})) \subset S_i \cup Y_1$ by induction hypothesis. The case $l_{i+1,1}(v) = z_{i+1}$ is impossible, because $z_{i+1} \notin \mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(V))$. The case $l_{i+1,1}(v) \in Y_1$ is impossible, too: Suppose $l_{i+1,1}(v) \in Y_1$, then $v <_L v_{i+1}$ by the linear constant restriction. But then $l_{i+1,2}(v) = h_{1,2}l_{i+1,1}(v)$ by induction hypothesis. And $l_{i+1,2}(v) = h_2\sigma_2(v) \in G_2$, which contradicts the choice of G_2. The case $l_{i+1,1}(v) = l_{i+1,1}(v_{i+1})$ is simple, τ_{i+1} is defined such that $l_{i+1,2}(v_{i+1}) = h_{1,2}l_{i+1,1}(v_{i+1})$. The case of $l_{i+1,1}(v) \in S_i$ was already handled above.
If $l_{i+1,2}(v) \in \mathrm{Sd}(z_{i+1})$ then $l_{i+1,2}(v) = z_{i+1}$ or $l_{i+1,2}(v) \in S_i$. $l_{i+1,2}(v) = l_{i+1,1}(v_{i+1})$ is impossible, because $l_{i+1,1}(v_{i+1}) \in C_1 \setminus Z_1$ and $l_{i+1,2}(v) \in C_2$. Similarly, $l_{i+1,2}(v) \notin Y_1$, because $l_{i+1,2}(v) \in C_2$ and $Y_1 \cap C_2 = \emptyset$. The case $l_{i+1,2}(v) = z_{i+1}$ is simple, because τ_{i+1} is defined such that $z_{i+1} = h_{1,2}l_{i+1,1}(v_{i+1})$. The case of $l_{i+1,2}(v) \in S_i$ was handled above. This shows Property (2).

Since $l_{i+1,1} = l_{i,1}$ we have $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(V)) = \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V)) \subset S_i \cup Y_1 \subset S_{i+1} \cup Y_1$. And $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(V)) = (\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(V)) \setminus \{y_{i+1}\}) \cup \{z_{i+1}\}$. Since $z_{i+1} \in S_{i+1}$, we have $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(V)) \subset S_{i+1} \cup Y_2$. This shows Property (3).

To show Property (4), we first note that since τ_{i+1} is the identity on S_i the property holds for all variables $v < v_{i+1}$ by induction hypothesis. Furthermore, since $\mathrm{Stab}^{\mathfrak{C}_1}(l_{i+1,1}(V)) = \mathrm{Stab}^{\mathfrak{C}_1}(l_{i,1}(V))$ the property also holds for all variables v with $Lab(v) = \Sigma_1$ which includes v_{i+1}. Remain the variables $v > v_{i+1}$ with $Lab(v) = \Sigma_2$. By induction hypothesis (3), $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v) \subset S_i \cup Y_2$. Since τ_{i+1} is the identity on S_i, there is only one interesting case, namely $y_{i+1} \in \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v))$. Otherwise, τ_{i+1} is the identity on $\mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v))$ and the property follows from the induction hypothesis. If $y_{i+1} \in \mathrm{Stab}^{\mathfrak{C}_2}(l_{i,2}(v))$ then $z_{i+1} \in \mathrm{Stab}^{\mathfrak{C}_2}(l_{i+1,2}(v))$ by definition of τ_{i+1}. But z_{i+1} is fibred with $l_{i+1,1}(v_{i+1})$ for which we already know that $\mathrm{Bottom}(\mathrm{Sd}(l_{i+1,1}(v_{i+1}))) \cap X = \emptyset$. Therefore $\mathrm{Bottom}(\mathrm{Sd}(l_{i+1,2}(v))) \cap X = \emptyset$, too.
This ends the induction proof.

By this induction, we showed that for $i = n$ there are assignments $l_{n,1}$ and $l_{n,2}$ such that $l_{n,1}$ is a restricted solution of $(\Gamma_{5,1}, L)$ and $l_{n,2}$ is a restrictive solution for $(\Gamma_{5,2}, L)$ and for every variable $v \in V$: $l_{n,2}(v) = h_{1,2}l_{n,1}(v)$. Therefore $l_{n,1}$ solves $\Gamma_{5,2}$ in $\mathfrak{C}_1^{\Sigma}$ and hence also Γ_4. It remains to show that the solution elements in $l_{n,1}$ are all ground in $(\mathfrak{C}_1^{\Sigma}, X)$. Let $v \in V$. If $Lab(v) = \Sigma_1$, then $\mathrm{Bottom}(\mathrm{Sd}(l_{n,1}(v))) \cap X = \emptyset$ by Property (4) of the induction. If $Lab(v) = \Sigma_2$, then $l_{n,1}(v) = h_{2,1}l_{n,2}(v)$ and $\mathrm{Bottom}(\mathrm{Sd}(l_{n,2}(v))) \cap X = \emptyset$. Therefore also $\mathrm{Bottom}(\mathrm{Sd}(l_{n,1}(v))) \cap X = \emptyset$. Thus for all $v \in V$ we have that $l_{n,1}(v)$ is ground in $(\mathfrak{C}_1^{\Sigma}, X)$ by Lemma 6.3.5. ■

Proposition 6.3.6 *If the input problem Γ_0 has a ground solution, then there exists a linear constant restriction L and an output pair $(\Gamma_{5,1}, \Gamma_{5,2})$ such that both $(\Gamma_{5,1}, L)$ and $(\Gamma_{5,2}, L)$ have a restrictive solution.*

The completeness proof is a very small extension of the completeness proof in the non-ground case.

Proof. Let σ be the ground solution in the combined solution domain $(\mathfrak{C}_1^{\Sigma}, X)$, and Z_1 the atom set of $C_1^{\Sigma_1}$. In Step 3, we identify two variables x and y, iff $\sigma(x) = \sigma(y)$. In Step 4, variable x receives label Σ_2, iff $\sigma(x) \in Z_1$. And the order $<$ is defined by: $x < y$, iff $\sigma(x) <_i \sigma(y)$ according to Definition 3.3.9. This gives the linear constant restriction L. Then σ obviously solves Γ_4 and therefore $\Gamma_{5,1}$ and $\Gamma_{5,2}$.

Because σ is ground, for every variable v: either $\sigma(v) \in Z_1$ or $h_{1,2}\sigma(v) \in Z_2$, but not both: Suppose both $\sigma(v) \in Z_1$ and $h_{1,2}\sigma(v) \in Z_2$. For every $z \in Z_1 \setminus X$: $h_{1,2}(z) \in C_2 \setminus Z_2$ and for every $z' \in Z_2 \setminus X$: $h_{2,1}(z') \in C_1 \setminus Z_1$ by definition of the fibring construction. Thus $\sigma(v) \in X$ and $h_{1,2}\sigma(v) \in X$ and $\sigma(v) = h_{1,2}\sigma(v)$ by definition of the fibring construction. But then $\sigma(v)$ is not ground.

Let $h_1 : \mathfrak{C}_1^{\Sigma_1} \to \mathfrak{A}_1^{\Sigma_1}$ be the qf-isomorphism between $\mathfrak{C}_1^{\Sigma_1}$ and $\mathfrak{A}_1^{\Sigma_1}$. Then $h_1\sigma$ solves $(\Gamma_{5,1}, L)$ in $\mathfrak{A}_1^{\Sigma_1}$ by the completeness proof in the general case. $h_1\sigma$ is restrictive, because if $Lab(v) = \Sigma_1$ then $\sigma(v) \in C_1 \setminus Z_1$ by the above, and qf-isomorphisms map non-atoms to non-atoms.

Let $h_2 : \mathfrak{C}_2^{\Sigma_2} \to \mathfrak{A}_2^{\Sigma_2}$ be the qf-isomorphism between $\mathfrak{C}_2^{\Sigma_2}$ and $\mathfrak{A}_2^{\Sigma_2}$. Then $h_2h_{1,2}\sigma$ solves $(\Gamma_{5,2}, L)$ in $\mathfrak{A}_2^{\Sigma_2}$ by the completeness proof in the general case. And $h_2h_{1,2}\sigma$ is restrictive, because if $Lab(v) = \Sigma_2$ then $\sigma(v) \in C_2 \setminus Z_2$ by the above, and qf-isomorphisms map non-atoms to non-atoms. ■

6.4 Is there a Logical Translation of Solving Problems with LCRs?

Unfortunately, the notion of a linear constant restriction is a purely technical, not very elegant one. Thus it would be nice, if we were able to translate it into a purely logical problem. But this task proves difficult, and it is indeed unlikely that there will be a general solution.

Definition 6.4.1 A formula φ is called *positive*, iff it is constructed by conjunction, disjunction, existential and universal quantification of atomic formulae, i.e., contains no negation or implication.
A formula ψ is called *negative*, iff ψ is equivalent to a formula $\neg\varphi$ where φ is a positive formula.

In [15], F. Baader and K. U. Schulz show that for positive formulae, the solving of problems with linear constant restrictions is equivalent to solving the full *positive* theory by proving that solving a formula with linear constant restriction is equivalent to solving the formula with an alternating quantifier prefix. We presented this statement as Lemma 3.2.29 in the section introducing quasi-free structures. The key insight there is that the atoms (of the linear constant restriction) play indeed the role of universal elements. But when under the scope of (an odd number of) negations, the atoms loose their universality property, they behave almost like any other element. This can be shown by the following

Proposition 6.4.2 *Let* $(\mathfrak{A}^\Sigma, X)$ *be a quasi-free structure and*

$$\gamma = \exists v_1 \ldots v_l \, \varphi(v_1, \ldots, v_l)$$

be a negative Σ*-sentence, where* φ *is a negative formula and* $v_1, \ldots, v_l$ *all free variables occurring in it. Then the following conditions are equivalent:*

1. $\mathfrak{A}^\Sigma \models \exists v_1 \ldots v_l \, \varphi(v_1, \ldots, v_l)$.
2. *There exist a* $k \leq \frac{l}{2}$ *and* $\vec{x}_1 \in \vec{X}, \vec{e}_1 \in \vec{A}, \ldots \vec{x}_k \in \vec{X}, \vec{e}_k \in \vec{A}$ *such that*
   (a) $|\vec{x}_1 \cup \ldots \cup \vec{x}_k \cup \vec{e}_1 \cup \ldots \cup \vec{e}_k| = l$,
   (b) $\mathfrak{A}^\Sigma \models \varphi(\vec{x}_1, \vec{e}_1 \ldots, \vec{x}_k, \vec{e}_k)$,
   (c) *all atoms in the sequences* $\vec{x}_1, \ldots, \vec{x}_k$ *are distinct,*
   (d) *for all* $j, 1 \leq j \leq k$*, the components of* $\vec{x}_j$ *are not contained in* $\mathrm{Stab}^{\mathfrak{A}}_{\Sigma}(\vec{e}_1) \cup \ldots \cup \mathrm{Stab}^{\mathfrak{A}}_{\Sigma}(\vec{e}_{j-1})$.

Proof. For the non-trivial part, let

$$\mathfrak{A}^\Sigma \models \exists v_1 \ldots v_l \, \varphi(v_1, \ldots, v_l).$$

Then there are a $k \leq \frac{l}{2}$ and elements $\vec{a}_1 \in \vec{A}, \vec{b}_1 \in \vec{A}, \ldots, \vec{a}_k \in \vec{A}, \vec{b}_k \in \vec{A}$ such that $|\vec{a}_1 \cup \ldots \cup \vec{a}_k \cup \vec{b}_1 \cup \ldots \cup \vec{b}_k| = l$ and

$$\mathfrak{A}^\Sigma \models \varphi(\vec{a}_1, \vec{b}_1, \ldots, \vec{a}_k, \vec{b}_k).$$

Let $\vec{x}_1 \in \vec{X}, \vec{y}_1 \in \vec{X}, \ldots, \vec{x}_k \in \vec{X}, \vec{y}_k \in \vec{X}$ be sequences of atoms such that all atoms in the sequence $\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k$ are different. The mapping h' that maps the sequences $\vec{x}_1$ to $\vec{a}_1$, $\vec{y}_1$ to $\vec{b}_1, \ldots, \vec{x}_k$ to $\vec{a}_k$, $\vec{y}_k$ to $\vec{b}_k$ can be extended to a surjective homomorphism h. Then

$$\mathfrak{A}^\Sigma \models \varphi(h(\vec{x}_1), h(\vec{y}_1), \ldots, h(\vec{x}_k), h(\vec{y}_k))$$

by assumption and therefore

$$\mathfrak{A}^\Sigma \models \varphi(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$$

since surjective homomorphisms preserve validity of positive formulae. This shows Condition (b). Conditions (c) and (d) follow immediately by the choice of the atoms all being different and the fact that each atom is its own stabiliser. ■

Unfortunately, this proposition also indicates that it is very unlikely to find a translation of problems with linear constant restrictions into problems expressed in purely logic terms in the general case. Consider an atom (from the linear constant restriction) that appears both in a positive and in a negative context. In the positive context, it would be translated into a universally quantified variable. In the negative context, the same variable had to be existentially quantified, which is obviously impossible. There are no quantifiers that behave sometimes this way sometimes another. And a general translation of the atom by a universally quantified variable is not correct.

6.5 A Stronger Combination Result?

We showed that the *existential* fragment of the free amalgamated product is decidable provided conjunctions of literals with linear constant restrictions are decidable in the components. A natural question to ask is, if we can decide a larger quantifier prefix fragment that just the existential one. This issue appears to be even more interesting, since F. Baader and K. U. Schulz [10] showed that for the case of purely positive constraints, the full positive theory, i.e., arbitrary quantifier prefixes, can be decided in the free amalgamated product. The answer we give in this section is a negative one. We will provide counterexamples from the field of equational unification for the general as well as for the ground case.

Undecidability of the Σ_3-Fragment in the General Case

Solvability of disunification problems with linear constant restrictions for the free theory and the theory of an AC-function symbol is shown to be decidable by F. Baader and K. U. Schulz in [9]. Therefore the existential theory of the free algebra of one AC-symbol and any finite number of free function symbols is decidable. Even better, the full first order theory of both the free theory [33, 70, 72] and the theory of an AC-function symbol and any number of constants [101] is decidable.

On the other hand, the Σ_3-fragment of the first order theory of an AC-function symbol, one free function symbol and one constant over the free algebra is shown to be undecidable by R. Treinen [113].

Proposition 6.5.1 *There exists an instance of combination, namely the combination of the theory of an AC-symbol and the free theory, the Σ_3-fragment of which is undecidable, though the full first order theories of both components are decidable.*

This result leaves a gap: What about the Σ_2-fragment? And indeed, for the theory of one AC-function, one free function and one constant over the free algebra, it is an open question whether or not the Σ_2-fragment is decidable. For the case of combination, a look at the ground case will show that there is no hope.

Undecidability of the Σ_2-Fragment in the Ground Case

The example in this subsection is given by the combination of the associative theory with the free theory. In [9], F. Baader and K. U. Schulz prove the decidability of the existential theory of the initial algebra of the signature consisting of one associative function symbol, one free function symbol and one constant. This result is obtained by as follows: The solvability of disunification problems with linear constant restrictions over the initial algebra is shown for the free theory and the associative theory. And one of the main results of that paper is the decidability of the existential fragment of the initial algebra of the combined signatures provided decidability of disunification problems with linear constant restrictions in the components.

On the other hand, R. Treinen ([113]) shows the *un*decidability of the Σ_2-fragment of the ground term algebra of the signature containing one associative function, one free function and one constant.

6.6 Rational Combination of Negative Constraints

In the previous sections, the combined solution domain was always the free amalgamated product. There exists another standard combined solution domain, namely rational amalgamation, as introduced in Chapter 5. So, one may expect that it is possible to solve negative constraints also in the rational amalgam. But unfortunately, this is not the case. We show that by means of a simple example, the combination of rational trees. Let signature $\Sigma = \{f\}$ and $\Delta = \{g\}$ where both f and g are unary and $X = \{x_1, x_2, x_3, \ldots\}$ be an infinite set of variables. Let $\mathfrak{R}(\Sigma, X)$ be the algebra of rational trees over f and $\mathfrak{R}(\Delta, X)$ the algebra of rational trees over g. Their rational amalgam $\mathfrak{R}(\Sigma, X) \odot \mathfrak{R}(\Delta, X)$ is isomorphic to $\mathfrak{R}(\Sigma \cup \Delta, X)$, the algebra of rational trees over f and g (see Theorem 5.4.2). Consider the following disunification problem $\Gamma = \{x \doteq f(z_1), z_1 \doteq g(z_2), z_2 \doteq f(z_3), z_3 \doteq g(x), x \not\doteq z_2, z_1 \not\doteq z_3\}$. This problem has a solution in the components, but it has no solution in the rational amalgam. In the labelling step of the decomposition algorithm, we choose the following indexing of the variables: $x \dot{\mapsto} \Sigma, z_2 \dot{\mapsto} \Sigma, z_1 \dot{\mapsto} \Delta, z_3 \dot{\mapsto} \Delta$. The subsystem $\Gamma_\Sigma = \{x \doteq f(z_1), z_2 \doteq f(z_3), x \not\doteq z_2, z_1 \not\doteq z_3\}$ has as one solution $\{x \mapsto f(x_1), z_1 \mapsto x_1, z_2 \mapsto f(x_2), z_3 \mapsto x_2\}$. Since $x_1 \neq x_2$, the disequations are clearly solved. The subsystem $\Gamma_\Delta = \{z_1 \doteq g(z_2), z_3 \doteq g(x), x \not\doteq z_2, z_1 \not\doteq z_3\}$ has as one solution $\{z_1 \mapsto g(x_1), z_2 \mapsto x_1, z_3 \mapsto g(x_2), x \mapsto x_2\}$. But Γ is unsolvable in $\mathfrak{R}(\Sigma \cup \Delta, X)$: Obviously, $x = f(g(f(g(x))))$, so x must be mapped to the infinite tree $fgfgfgfg\ldots$. And $z_2 = f(g(f(g(z_2))))$, so z_2 must also be mapped to the infinite tree $fgfgfgfg\ldots$. But then, the disequation $x \not\doteq z_2$ is violated.

It is of course simple to regain soundness of the decomposition algorithm by severely restricting the admissible solutions in the components. But then, one faces completeness problems. There seems to be no general way out of this dilemma.

Part II

The Independence of Negative Constraints Property

The constraint problems we consider in this chapter are conjunctions of literals. Clearly, every such constraint problem Γ can be written as $\Gamma^+ \wedge \bigwedge_{i=1}^{k} C_i^-$ where Γ^+ is the conjunction of all positive constraints (atoms) of Γ and $\bigwedge_{i=1}^{k} C_i^-$ is the conjunction of all negative constraints (negated atoms) of Γ. If we do not need to refer to specific negative constraints, we abbreviate $\bigwedge_{i=1}^{k} C_i^-$ by Γ^-.

Definition A theory has the *independence of negative constraints* property, iff for every constraint problem Γ we have:

> Γ is solvable, iff
> for every $i = 1, \ldots, k$ the conjunction $\Gamma^+ \wedge C_i^-$ is solvable.

6.7 Independence Properties of Equational Theories

In equational theories, the only negative constraints are of course disequations. For a general introduction to disunification theory, we refer the reader to [31, 32, 33]. It is easy to see that we can restrict our attention to those cases where the only disequations occurring are disequations between variables. Because if $s \not\doteq t$ is a disequation and x and y are new variables, then $s \not\doteq t$ is equivalent to $x \doteq s, y \doteq t, x \not\doteq y$. Remember that we write $\mu U(\Gamma^+)$ for the minimal complete set of unifiers of a unification problem Γ^+, if this set exists.

Lemma 6.7.1 *Let E be an equational theory of unification type different from 0. Let Γ be a disunification problem where Γ^- contains only disequations between variables occurring in Γ^+. If Γ has a solution σ, then one substitution τ from the minimal complete set of unifiers of Γ^+ more general than σ is a solution of Γ.*

Proof. Let $\mu U(\Gamma^+)$ be the minimal complete set of unifiers for Γ^+ and σ a solution of Γ. Since σ is a solution of Γ^+, there must be a $\tau \in \mu U(\Gamma^+)$ such that σ is an instance of τ. Now for any two variables $x, y \in \mathit{Var}(\Gamma^+)$: if $\tau(x) =_E \tau(y)$ then $\sigma(x) =_E \sigma(y)$. Thus if $\sigma(x) \neq_E \sigma(y)$ then $\tau(x) \neq_E \tau(y)$. Therefore τ is a solution of Γ^-. ■

In the above lemma, if Γ^- contains variables not occurring in Γ^+, then these variables are only constrained to be mapped to solution elements different from other variables in Γ^-. Thus these types of disequations only impose restrictions on the minimal size of a finite solution domain. But since all our solution domains are infinite, disequations between variables not occurring in Γ^+ do not impose any restrictions at all here.

Unitary Theories

Theorem 6.7.2 *Let E be a unitary equational theory. Then E has the independence of negative constraints property.*

Proof. This is an immediate consequence of Lemma 6.7.1. Let for $i \leq k$ the problem $\Gamma^+ \wedge C_i^-$ have a solution. Then for all $i \leq k$: the mgu μ of Γ^+ is a solution of $\Gamma^+ \wedge C_i^-$. Thus μ is a solution of $\Gamma^+ \wedge C_1^- \wedge \ldots \wedge C_k^-$. ∎

In a later subsection, we will show that the inverse (i.e., independence property entailing E being unitary) is *not* true.

Equational Theories without Independence Property

In this section, we prove the following statement: If an equational theory is such that one of its unification problems has a minimal complete set of unifiers of finite cardinality, but no single most general unifier, then this theory does not have the independence property. The incomparability of the solutions is used to construct disequations that can be solved individually, but not collectively.

Lemma 6.7.3 *Let E be a non-unitary equational theory. Let Γ be a unification problem with variables X and μ_1 and μ_2 be two solutions of Γ with domain X such that neither μ_1 is more general than μ_2 nor vice versa. Then there exists a disequation for which μ_1 is not a solution, but μ_2 is.*

Proof. We can assume that both μ_1 and μ_2 are idempotent.
Now μ_1 is not more general than μ_2, that is
there is no substitution λ such that $\mu_2(x) =_E \lambda \circ \mu_1(x)$ for every $x \in X$, i.e.
for every substitution λ there is an $x \in X$ such that $\mu_2(x) \neq_E \lambda \circ \mu_1(x)$.
In particular, if we choose λ to be μ_2 we get: There is an $x \in X$ such that $\mu_2(x) \neq_E \mu_2 \circ \mu_1(x)$.
Now consider the disequation $x \not\doteq \mu_1(x)$. The above shows that μ_2 is a solution.
But since μ_1 is idempotent, clearly $\mu_1(x) = \mu_1 \circ \mu_1(x)$; thus μ_1 is no solution. ∎

This section's claim is now a simple consequence.

Lemma 6.7.4 *Let E be a non-unitary equational theory. Let Γ be a unification problem with variables X and $\{\mu_1, \ldots, \mu_n\}$ (where $n > 1$) be a minimal complete set of solutions of Γ with domain X. Then there exists a set of disequations $\{d_1, \ldots, d_n\}$ such that for each $i = 1, \ldots, n$ the problem $\Gamma \wedge d_i$ is solvable, but $\Gamma \wedge \{d_1, \ldots, d_n\}$ is not.*

Proof. Again, we can assume that the solutions $\mu_1, \ldots, \mu_n$ are idempotent. For each $i = 1, \ldots, n$ consider the solutions μ_i and some μ_j where $j \neq i$. By the above lemma, there is an $x \in X$ such that μ_j solves $x \neq \mu_i(x)$, but μ_i does not. Define d_i to be $x \neq \mu_i(x)$. Then $\Gamma \wedge d_i$ is solvable, μ_j is a solution. But $\Gamma \wedge \{d_1, \ldots, d_n\}$ is not, because every solution of it is an instance of some μ_i and each d_i excludes μ_i (and its instances) from the set of solutions. ■

Theorem 6.7.5 *Let E be an equational theory that has a unification problem where the cardinality of the minimal complete set of unifiers is finite and larger than 1. Then E has* not *the independence property.*

Corollary 6.7.6 *Let E be a finitary equational theory. Then E has* not *the independence property.*
The theory A of an associative function symbol and constants has not *the independence property.*

For the second statement, consider the following A-unification problem with constants, written as a word unification problem: $axb \doteq yybb$ where a and b are constants and x and y are variables. It has the following two most general solutions: $\{(x \mapsto ab, y \mapsto a), (x \mapsto zazb, y \mapsto az)\}$ where z is a new variable ranging over (nonempty) words over a and b. It is trivial to check that the two substitutions are solutions. To see that they they form a minimal complete set, consider the way the Plotkin algorithm [85] would solve the unification problem. Clearly, y must either be mapped to a, which immediately gives the first solution, or a is a proper prefix of the solution for y, which gives rise to the second solution. The solution for x is in both cases always immediately determined by the solution for y.

Commutative Theories have the Independence Property

In this section, we present a whole class of equational theories, the so-called commutative theories, which have the independence property. Commutative theories were studied independently by Franz Baader [4] and Werner Nutt [79] who calls them "monoidal theories". Amongst these theories, there is the theory *AMh* of Abelian monoids with a homomorphism, which is not unitary. This shows that the property of being of unification type unitary and the independence property are not equivalent.

The following heavily draws from results by F. Baader on commutative theories in [4]. The results about unification (i.e. no disequations) can be found there. The proposition that commutative theories have the independence property (for elementary unification) is new, a fruit of cooperation with F. Baader.

W. Nutt's definition of a monoidal theory is based on the signature and therefore easier to grasp then F. Baader's category-theoretical definition. Thus we give it first and provide some examples.

Let E be a set of identities. Then E (and the equational theory defined by E) is called *monoidal*, iff it satisfies the following properties:

1. The signature of E contains one binary function symbol $\cdot$, one nullary function symbol 1, and all other function symbols are unary.
2. The symbol $\cdot$ is associative and commutative, i.e., $((x \cdot y) \cdot z) =_E (x \cdot (y \cdot z))$ and $(x \cdot y) =_E (y \cdot x)$ hold.
3. The symbol 1 is a unit for $\cdot$, i.e., $(x \cdot 1) =_E x$ holds.
4. Every unary function symbol h is a homomorphism for $\cdot$ and 1, i.e., $h(x \cdot y) =_E h(x) \cdot h(y)$ and $h(1) =_E 1$ hold.

The list of examples that follows is taken from [4].

Example 6.7.7 We consider the following signatures:
$\Sigma_1 := \{\cdot, 1\}$ where $\cdot$ is binary and 1 is nullary.
$\Sigma_2 := \Sigma_1 \cup \{^{-1}\}$ and $\Sigma_3 := \Sigma_1 \cup \{h\}$ where $^{-1}$ and h are unary.
$\Sigma_4 := \Sigma_2 \cup \Sigma_3$.

1. The theory of Abelian monoids.
   The signature is Σ_1 and $AM := \{x \cdot (y \cdot z) = (x \cdot y) \cdot z, x \cdot y = y \cdot x, x \cdot 1 = x\}$.
2. The theory *AIM* of idempotent Abelian monoids.
   The signature is Σ_1 and $AIM := AM \cup \{x \cdot x = x\}$.
3. The theory *AMh* of Abelian monoids with one homomorphism.
   The signature is Σ_3 and $AMh := AM \cup \{h(x) \cdot h(y) = h(x \cdot y), h(1) = 1\}$.
4. The theory *AIMh* of idempotent Abelian monoids with one homomorphism. The signature is Σ_3 and $AIMh := AIM \cup \{h(x) \cdot h(y) = h(x \cdot y), h(1) = 1\}$.
5. The theory *AMi* of Abelian monoids with an involution.
   The signature is Σ_3 and $AMi := AM \cup \{h(x) \cdot h(y) = h(x \cdot y), h(h(x)) = x\}$.
6. The theory *AIMi* of idempotent Abelian monoids with an involution.
   The signature is Σ_3 and $AIMi := AMi \cup \{x \cdot x = x\}$.
7. The theory AG_m of Abelian groups of exponent m ($m \in \mathbb{N}$). The signature is Σ_2 and $AG_m := AM \cup \{x \cdot x^{-1} = 1, x^m = 1\}$. $AG = AG_0$ is the theory of Abelian groups.
8. The theory *AGi* of Abelian groups with an involution.
   The signature is Σ_4 and $AGi := AG \cup AMi$.
9. The theory *AGh* of Abelian groups with a homomorphism.
   The signature is Σ_4 and $AGh := AG \cup \{h(x) \cdot h(y) = h(x \cdot y), h(1) = 1\}$.

For the following discussion we need to introduce at least some of the category theory terminology of [4]. Of course, we cannot give an introduction to category theory here. In short, a category is a collection of objects and a collection of morphisms between these objects such that composition of morphisms is associative and each object has a unit morphism, the identity map of the object. An object A is called a zero object, if for each object B there exists a unique morphism from A to B and a unique morphism from B to A. A product of two objects A and B consists of an object $A\times B$ and two morphisms $\pi_A : A\times B \to A$ and $\pi_B : A \times B \to B$ such that for every other object D and morphisms $d_A : D \to A$ and $d_B : D \to B$ there exists a unique morphism $d! : D \to A \times B$ with $d_A = \pi_A \circ d!$ and $d_B = \pi_B \circ d!$. The coproduct is the dual of the product. Therefore a coproduct of two objects A and B consists of an object $A+B$ and two morphisms $i_A : A \to A + B$ and $i_B : B \to A + B$ such that for every other object E and morphisms $e_A : A \to E$ and $e_B : B \to E$ there exists a unique morphism $e! : A + B \to E$ with $e_A = e! \circ i_A$ and $e_B = e! \circ i_B$. For more information, see [83] for example.

If E is an equational theory, and X a set of variables, then let $F_E(X)$ be the free algebra over the variety of E with generators X.

Let $\Gamma = \langle s_i \doteq t_i; 1 \leq i \leq n\rangle_E$ be an E-unification problem and X be the finite set of variables x occurring in some s_i or t_i. Evidently, we can consider s_i and t_i as elements of $F_E(X)$. Since we do not distinguish between $=_E$-equivalent unifiers, any E-unifier of Γ can be regarded as a homomorphism of $F_E(X)$ into $F_E(Y)$ for some finite set Y (of variables). Let $I = \{x_1, \ldots, x_n\}$ be a set of cardinality n. We define homomorphisms

$$\sigma, \tau : F_E(I) \to F_E(X) \text{ by } \sigma(x_i) := s_i \text{ and } \tau(x_i) := t_i \quad (i = 1, \ldots, n).$$

Now $\delta : F_E(X) \to F_E(Y)$ is an E-unifier of Γ, iff $\delta(\sigma(x_i)) = \delta(s_i) = \delta(t_i) = \delta(\tau(x_i))$ for $i = 1, \ldots, n$, i.e., iff $\delta\sigma = \delta\tau$. Thus an E-unification problem can be written as a pair $\langle \sigma = \tau\rangle_E$ of morphisms $\sigma, \tau : F_E(I) \to F_E(X)$ in the following category:

Definition 6.7.8 ([4]:3.1) Let E be an equational theory and V be a denumerable set. Then the category $C(E)$ is defined as follows:

- The objects of $C(E)$ are the algebras $F_E(X)$ for finite subsets X of V. We denote the class of these objects by $F(E)$.
- The morphisms of $C(E)$ are the homomorphisms between these objects.
- The composition of morphisms is the usual composition of mappings.

For morphisms $\sigma : F_E(X) \to F_E(Y), \gamma : F_E(X) \to F_E(Z)$ we have $\gamma \leq_E \sigma$, iff there is a morphism $\lambda : F_E(Z) \to F_E(Y)$ such that $\sigma = \lambda\gamma$.

Definition 6.7.9 An equational theory E is *commutative*, iff its category $C(E)$ is semiadditive, i.e., $C(E)$ has a zero object, every pair of objects has a coproduct, and product and coproduct coincide.

From now on, let E be a commutative theory.

Lemma 6.7.10 ([4]:6.2) *Let $\Gamma = \langle s_j \doteq t_j\rangle_E$ be an elementary E-unification problem and let $\{\sigma_1, \ldots, \sigma_n\}$ be a finite complete set of E-unifiers of Γ. Then there exists an E-unifier σ of Γ such that the singleton $\{\sigma\}$ is a complete set of E-unifiers of Γ.*

Lemma 6.7.11 ([4]:6.3) *Let* $\Gamma = \langle s_j \doteq t_j \rangle_E$ *be an elementary E-unification problem and let* $U = \{\sigma_1, \sigma_2, \sigma_3, \ldots\}$ *be an infinite set of E-unifiers such that the* σ_i *do not lie (w. r. t.* $\leq_E$*) above a single E-unifier of* Γ*. Then there does not exist a minimal complete set* $\mu U(\Gamma)$*.*

The proofs can be found in [4].

Lemma 6.7.12 ([4]:9.3) *The theory AMh is of unification type zero.*

Thus for elementary unification, every unification problem of the theory *AMh* either has at most one most general unifier or no minimal complete set of unifiers. An example of a problem with no minimal complete set of unifiers is $\langle h(x_1)h(x_2) \doteq x_2 h(h(x_3))\rangle_{AMh}$. The solutions have the form $x_1 \mapsto z, x_2 \mapsto h(z) \cdot h^2(z) \cdot \ldots \cdot h^{n+1}(z), x_3 \mapsto h^n(z)$ for each $n \geq 0$.

Theorem 6.7.13 *For elementary disunification problems, commutative theories have the independence of negative constraints property.*

Proof. This proof follows closely the one of Lemma 6.2 in [4]. Let E be a commutative theory. Let $\Gamma = \langle \sigma \doteq \tau, \langle x_i \not\doteq y_i \rangle_{1 \leq i \leq k} \rangle_E$ be an elementary disunification problem where $x_i, y_i \in X$. Clearly, if Γ has a solution γ, then γ is a solution of $\Gamma_i := \langle \sigma \doteq \tau, x_i \not\doteq y_i \rangle_E$ for $1 \leq i \leq k$. For the non-trivial direction, let γ_i be a solution of Γ_i for all $1 \leq i \leq n$. Then each γ_i is a solution of $\langle \sigma = \tau \rangle_E$. We show that there is an E-unifier γ of $\langle \sigma = \tau \rangle_E$ which is more general then each γ_i. We have $\sigma, \tau : F_E(I) \to F_E(X)$ and $\gamma_i : F_E(X) \to F_E(Y_i)$. With $Y := Y_1 \uplus \ldots \uplus Y_k$, $F_E(Y)$ is the coproduct and product of the $F_E(Y_i)$. Let $\pi_1, \ldots, \pi_k$ be the corresponding projections. Then there exists a unique morphism $\gamma : F_E(X) \to F_E(Y)$ such that $\gamma_i = \pi_i \gamma$ for $1 \leq i \leq k$. The morphism γ is an E-unifier of $\langle \sigma = \tau \rangle_E$, since $\gamma\sigma = \gamma\tau$ iff $\pi_i\gamma\sigma = \pi_i\gamma\tau$ for $i = 1, \ldots, k$ (by definition of product). And γ is more general then each γ_i, since $\gamma_i = \pi_i\gamma$ by definition.

Therefore for all $x, y \in X$, if $\gamma(x) = \gamma(y)$, then $\gamma_i(x) = \gamma_i(y)$. By contraposition, for all $x, y \in X$ if $\gamma_i(x) \neq \gamma_i(y)$, then $\gamma(x) \neq \gamma(y)$. Thus from $\gamma_i(x_i) \neq \gamma_i(y_i)$ follows $\gamma(x_i) \neq \gamma(y_i)$, and γ solves $\langle x_i \not\doteq y_i \rangle_{1 \leq i \leq k}$. This shows that γ is a solution of Γ. ∎

Corollary 6.7.14 *There exists an equational theory, namely AMh, which has the independence property, but is not unitary.*

6.8 Combining Equational Theories and the Independence Property

In this section, we present conditions under which the independence property of equational theories is preserved under combination. These conditions are quite restrictive as the following theorem shows.

Theorem 6.8.1 *Let E and F be two unitary regular and collapse-free theories over disjoint signatures. Then the combined theory $E \cup F$ is again unitary, regular, and collapse-free.*

Regularity and collapse-freeness are properties of the axioms of E and F. If the signatures are disjoint, then theses properties are clearly preserved in the union $E \cup F$. Thus the main statement of the theorem is that the combination of two unitary regular and collapse-free theories is again unitary. Unfortunately, the requirement that E and F be regular and collapse-free cannot be weakened as the following example shows. Consider the theory of Boolean rings, which is neither regular nor collapse-free. It is known (see [17]) that in this theory, unification with constants is unitary. But general unification is finitary. And in any theory, general unification can be regarded as an instance of combining unification with constants in that theory with syntactic unification.

The above demanded requirements are rather restrictive. There are only two "natural" theories which come to mind that are both unitary and regular and collapse-free. One is of course syntactic unification. And the other is single-sided distributivity, such as distributivity to the left ($D_L = \{f(g(x,y),z) = g(f(x,z), f(y,z))\}$) and distributivity to the right.

We will not prove the theorem here, because it is a special case of Theorem 6.10.17 that we present later. A direct proof of the above theorem would involve complicated rewriting methods. The introduction of an enormous technical apparatus only to see later that we can prove the theorem in a more general way seems a waste. The proof of the general theorem uses clear algebraic methods and is shorter than a specific proof can be.

6.9 Independence Properties of Quasi-free Structures

6.9.1 Generalising Basic Notions of Unification Theory

Before we can start to extend the results of the previous sections to quasi-free structures, it is necessary to generalise some basic notions of unification theory to quasi-free structures.

Remember that our constraint problems are existentially quantified conjunctions of literals.

Definition 6.9.1 Let Γ be a positive constraint problem of the quasi-free structure $(\mathfrak{A}^{\Sigma}, X)$. Let σ and τ be two solutions of Γ. We say σ is *more general* than τ with respect to the variables in Γ (and write $\sigma \leq_\Gamma \tau$), iff there exists an endomorphism $m \in End_{\mathfrak{A}}^{\Sigma}$ such that m is the identity on $X \setminus \mathrm{Stab}^{\mathfrak{A}}(\mathrm{rng}(\sigma))$ and for all $x \in Var(\Gamma)$ holds $m \circ \sigma(x) = \tau(x)$.
A set S of solutions for Γ is called *complete*, iff for every solution τ of Γ there exists a solution $\sigma \in S$ with $\sigma \leq_\Gamma \tau$.

A set S of solutions for Γ is called *minimal*, iff for all $\sigma, \sigma' \in S : \sigma \leq_\Gamma \sigma' \Longrightarrow \sigma = \sigma'$.

With these notions, we can define the solution type of a quasi-free structure.

Definition 6.9.2 A quasi-free structure $(\mathfrak{A}^\Sigma, X)$ is called

unitary, iff for every positive constraint problem a minimal complete set of solutions exists and has cardinality at most one.

finitary, iff for every positive constraint problem a minimal complete set of solutions exists and has finite cardinality.

infinitary, iff every positive constraint problem has a minimal complete set of solutions (but this set may be infinite).

of type 0, iff there is a positive constraint problem that has no minimal complete set of solutions.

Remember that by Definition 5.2.5 a quasi-free structure $(\mathfrak{A}^\Sigma, X)$ is called non-collapsing, iff every endomorphism $m \in End_{\mathfrak{A}}^{\Sigma}$ maps non-atoms to non-atoms. In order to define the notion *regular* for quasi-free structures, we start with the following observation, which is a generalisation of Lemma 5.2.4.

Lemma 6.9.3 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure, $m \in End_{\mathfrak{A}}^{\Sigma}$ an endomorphism, and $a \in A$ some element. Suppose $\mathrm{Stab}^{\mathfrak{A}}(a) = \{x_1, \ldots x_k\}$. Then $\mathrm{Stab}^{\mathfrak{A}}(m(a)) \subseteq \bigcup_{i=1}^{k} \mathrm{Stab}^{\mathfrak{A}}(m(x_i)) = \mathrm{Stab}^{\mathfrak{A}}(m(\mathrm{Stab}^{\mathfrak{A}}(a)))$.*

Proof. Let $m_1, m_2 \in End_{\mathfrak{A}}^{\Sigma}$ be two endomorphisms such that they coincide on $\bigcup_{i=1}^{k} \mathrm{Stab}^{\mathfrak{A}}(m(x_i))$. We show they coincide on $m(a)$. If m_1 and m_2 coincide on $\bigcup_{i=1}^{k} \mathrm{Stab}^{\mathfrak{A}}(m(x_i))$, then they coincide on $\{m(x_1), \ldots, m(x_k)\}$. Then $m_1 \circ m$ and $m_2 \circ m$ coincide on $\{x_1, \ldots, x_k\} = \mathrm{Stab}^{\mathfrak{A}}(a)$, hence $m_1 \circ m(a) = m_2 \circ m(a)$, in other words, m_1 and m_2 coincide on $m(a)$. ∎

Definition 6.9.4 A quasi-free structure $(\mathfrak{A}^\Sigma, X)$ is *regular*, iff for all $m \in End_{\mathfrak{A}}^{\Sigma}$ and all $a \in A : \mathrm{Stab}^{\mathfrak{A}}(m(a)) = \mathrm{Stab}^{\mathfrak{A}}(m(\mathrm{Stab}^{\mathfrak{A}}(a)))$.

6.9.2 Unitary Quasi-free Structures

Lemma 6.9.5 *Let $(\mathfrak{A}^\Sigma, X)$ be a quasi-free structure of solution type different from 0. Let $\Gamma = \Gamma^+ \wedge \Gamma^-$ be a constraint problem. If Γ has a solution τ, then one solution σ from the minimal complete set of solutions of Γ^+ more general than τ is a solution of Γ.*

Proof. Let $\mu S(\Gamma^+)$ be the minimal complete set of solutions of Γ^+ and τ a solution of Γ. Since τ is a solution of Γ^+, there exists a $\sigma \in \mu S(\Gamma^+)$ with $\sigma \leq \tau$, i.e., there is an endomorphism $m \in End_{\mathfrak{A}}^{\Sigma}$ with $m\sigma(x) = \tau(x)$ for all $x \in Var(\Gamma)$. Because endomorphisms preserve negative constraints in countercurrent direction, σ is a solution of Γ^-. ■

Theorem 6.9.6 *Let* $(\mathfrak{A}^\Sigma, X)$ *be a unitary quasi-free structure. Then* $(\mathfrak{A}^\Sigma, X)$ *has the independence property.*

Proof. This is an immediate consequence of the above lemma. Let $\Gamma = \Gamma^+ \wedge \bigwedge_{i=1}^{k} C_i^-$ and for all $1 \leq i \leq k$ the problem $\Gamma^+ \wedge C_i^-$ have a solution. Then for all $1 \leq i \leq k$: the most general solution μ of Γ^+ is a solution of C_i^-. Thus μ solves Γ. ■

It is worth mentioning that G. Smolka and R. Treinen [104] give an example of a quasi-free structure which is not an equational theory, is of type infinitary and has the independence property, namely their feature trees with arity.

6.10 Combining Quasi-free Structures and the Independence Property

This section's aim is to lift the modularity result for equational theories (6.8.1) to the more general case of quasi-free structures. The theorem to be shown therefore reads as follows.

Theorem 6.10.1 *Let* $(\mathfrak{A}^\Sigma, X)$ *and* $(\mathfrak{B}^\Delta, X)$ *be two unitary regular non-collapsing quasi-free structures over disjoint signatures. Then the free amalgamated product* $\mathfrak{A}^\Sigma \otimes \mathfrak{B}^\Delta$ *is again unitary, regular and non-collapsing.*

That the free amalgamated product is regular and non-collapsing is again a simple consequence of the fact that the components are, and that the signatures are disjoint. Endomorphisms in the free amalgamated product are pairs of endomorphisms of the component structures. If both components are regular and non-collapsing, the so is the pair, signature disjointness presupposed.

In this section, we will refer frequently to [94]. The paper contains a deterministic combination algorithm for unitary regular collapse-free theories. We show here that the algorithm computes a most general solutions.

6.10.1 $\mathcal{L}$-convex Theories and Deterministic Combination

In this subsection, we will briefly recall all relevant definitions and results about $\mathcal{L}$-convex theories and deterministic combination as presented in Sections 4.3 and 4.4 of [94]. The notions of decision sets and generalised linear constant restrictions were defined in our Section 4.3 in the chapter on optimisation techniques. It is important to note that what is called a "generalised linear constant restriction" in [94], is called here a "complete decision set". Let $\mathcal{U}$ be a set of variables. For us, a generalised linear constant restriction is not a set of constraints, but a triple $(\Pi, Lab, <)$ where Π is a partition of the variables $\mathcal{U}$, Lab a labelling function, and $<$ a partial order on $\mathcal{U}$.

Although not stated explicitly, the deterministic combination algorithm presented by K. U. Schulz in [94] is not just designed for the combination of equational theories, but also for combining quasi-free structures. In the deterministic algorithm, one just has to replace all references to equational theories by quasi-free structures. And the proofs of the propositions that we will subsequently present in their general form are still valid.

Proposition 6.10.2 *Let Γ be a $\Sigma \cup \Delta$-constraint problem in decomposed form $\Gamma_\Sigma \cup \Gamma_\Delta \cup \Gamma_{\neq}$. Let $\mathcal{U} = \mathrm{Var}(\Gamma)$. Then Γ is solvable if and only if there exists a complete decision set C_L on $\mathcal{U}$, where $\Gamma_{\neq} \subset C_L$, such that the Σ-constraint problem with decision set $(\Gamma_\Sigma \cup \Gamma_{\neq}, C_L)$ is solvable and the Δ-constraint problem $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$ is solvable.*

Proof. First assume that the $\Sigma \cup \Delta$-constraint problem Γ has a solution. By Proposition 6.2.13 there exists a linear constant restriction L such that $(\Gamma_{5,\Sigma}, L)$ has a solution σ_Σ in $(\mathfrak{A}^\Sigma, X)$ and $(\Gamma_{5,\Delta}, L)$ has a solution σ_Δ in $(\mathfrak{B}^\Delta, X)$. We use the linear constant restriction and the variable identification with representation function ρ that maps each variable to its representative in Step 3 of Algorithm 6.2.4 to construct a complete decision set. Let

$$\begin{aligned} C_= &:= \{x \doteq y \mid \rho(x) = \rho(y)\} \cup \{x \not\doteq y \mid \rho(x) \neq \rho(y)\}, \\ C_{Lab} &:= \{x \dot{\mapsto} \Sigma \mid Lab(\rho(x)) = \Sigma\} \cup \{x \dot{\mapsto} \Delta \mid Lab(\rho(x)) = \Delta\}, \\ C_< &:= \{x \dot{<} y \mid \rho(x) <_L \rho(y)\}, \end{aligned}$$

and let C_L be the closure of $C_= \cup C_{Lab} \cup C_<$. Note that $\Gamma_{\neq} \subset C_=$. And C_L is clearly complete. Since the only difference between $\Gamma_{5,\Sigma}$ and $\Gamma_\Sigma \wedge \Gamma_{\neq}$ is the variable identification, $\sigma_\Sigma \circ \rho$ is a solution of $\Gamma_\Sigma \wedge \Gamma_{\neq}$. And $\sigma_\Sigma \circ \rho$ respects C_L by definition of C_L. Hence $\sigma_\Sigma \circ \rho$ solves $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_L)$. Analogously, $\sigma_\Delta \circ \rho$ solves $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$.

For the inverse direction, assume there exists a complete decision set C_L with $\Gamma_{\neq} \subset C_L$ such that $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_L)$ has solution σ_Σ and $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$ has solution σ_Δ. The equation decisions in C_L form an equivalence relation. Choose a representant for each equivalence class and let ρ be the function that maps each variable to its representant. Define a linear constant restriction $L = (Lab, <)$ as follows. For each labelling decision $x \dot{\mapsto} \Sigma$ (resp. $x \dot{\mapsto} \Delta$) set $Lab(\rho(x)) = \Sigma$ (resp. Δ). Choose an arbitrary extension of the partial order $\{\rho(x) < \rho(y) \mid x \dot{<} y \in C_L\}$ to a linear order $<$. Since the only difference between $\Gamma_{5,\Sigma}$ and $\Gamma_\Sigma \wedge \Gamma_{\neq}$ is the variable identification, σ_Σ is a solution of $\Gamma_{5,\Sigma}$, even if not all variables in the domain of σ_Σ appear in $\Gamma_{5,\Sigma}$ due to the identification. The definition of Lab ensures that the labelling informations of C_L and L are the same modulo variable identification. The ordering decisions in C_L do not necessarily form a linear order. But by definition of a complete set of decisions, each two variables with different labels are ordered. Hence any linear extension of the ordering information will give rise to the same constant restrictions, as explained in Section 4.2. And σ_Σ respects the ordering decisions. Therefore σ_Σ solves $(\Gamma_{5,\Sigma}, L)$, and, analogously, σ_Δ solves $(\Gamma_{5,\Delta}, L)$. By Proposition 6.2.5, Γ has a solution. ∎

Corollary 6.10.3 *Let $\sigma = \sigma_\Sigma \otimes \sigma_\Delta$ as in Definition 6.2.11. Then σ is a solution of Γ, iff σ_Σ is a solution of $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_L)$ and σ_Δ is a solution of $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$.*

Definition 6.10.4 Let $\mathcal{U}$ be a set of variables. A complete decision set $C_L \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ is called a *faithful* extension of $C \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$, if $C \subseteq C_L$ and if C and C_L have the same set of equality decisions.

Lemma 6.10.5 *Let $C \neq C_\top$[3] be a decision set in $\mathcal{C}_{\mathcal{L}(\mathcal{U})}$. Then C has a faithful extension to a complete decision set $C_L \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$.*

The proof is given in [94].

Definition 6.10.6 The decision set $C_2 \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ is a *cover point* for the constraint problem with decision set (Γ, C_1), if $C_1 \subseteq C_2$, and for each complete decision set $C_L \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ that faithfully extends C_2 there exists a solution of (Γ, C_L).
The cover point C_2 of (Γ, C_1) is called a *universal* cover point for (Γ, C_1), if $C_2 \subseteq C_L$ for all complete decision sets $C_L \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ where $C_1 \subseteq C_L$ and the constraint problem with decision set (Γ, C_L) is solvable.

Definition 6.10.7 The quasi-free structure $(\mathfrak{A}^\Sigma, X)$ is *$\mathcal{L}$-convex*, iff for every constraint problem with decision set (Γ, C) there exists a universal cover point.

$(\mathfrak{A}^\Sigma, X)$ is *effectively* $\mathcal{L}$-convex, if there exists an algorithm that computes a universal cover point for each constraint problem with decision set (Γ, C).

Let $(\mathfrak{A}^\Sigma, X)$ and $(\mathfrak{B}^\Delta, X)$ denote two effectively $\mathcal{L}$-convex quasi-free structures over disjoint signatures Σ and Δ respectively. We shall now give a deterministic combination algorithm that may be used to decide solvability of $\Sigma \cup \Delta$-constraint problems.

Deterministic Combination Algorithm

The input of the algorithm is a $\Sigma \cup \Delta$-constraint problem Γ. We may assume that Γ is in decomposed form $\Gamma_\Sigma \wedge \Gamma_\Delta \wedge \Gamma_{\neq}$ where $\Gamma_{\neq}$ is the set of all disequations between variables. Steps 1 and 2 of the Decomposition algorithm 6.2.4 show

[3] Remember that in the text following Definition 4.3.4, $C_\top$ was defined as the closure of any inconsistent decision set.

that each constraint problem can be transformed into one in decomposed form by a simple preprocessing step. Let $\mathcal{U} := Var(\Gamma)$ and $n := |\mathcal{U}|$. The algorithm is organised in a series of rounds. Each round has a pure Σ (respectively Δ) constraint problem (Γ_I, C) with decision set C, where $I \in \{\Sigma, \Delta\}$ and $C \neq C_\top$. The input for the first round is (Γ_Σ, C_1), where $C_1 := C_\bot = Clo(\emptyset)$.

Round 1: We compute a universal cover point $C_2 \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ for (Γ_Σ, C_1) with respect to $(\mathfrak{A}^\Sigma, X)$. If $C_2 = C_\top$, then we stop with failure. In the other case, (Γ_Δ, C_2) is the input for round 2.

Round $k \geq 2$: Assume that the input for this round is (Γ_I, C_k), where $I \in \{\Sigma, \Delta\}$. We compute a universal cover point $C_{k+1} \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ for (Γ_I, C_k) with respect to the given quasi-free structure. The algorithm stops in two cases:

(i) If $C_{k+1} = C_\top$, then we stop with failure.

(ii) In the other case, if C_{k+1} and C_k have the same set of equality constraints, then we stop with success.

In the remaining case, the J-constraint problem with decision set (Γ_J, C_{k+1}), where $\{I, J\} = \{\Sigma, \Delta\}$, is the input for round $k + 1$.

Proposition 6.10.8 *The Deterministic Combination Algorithm terminates after at most $n + 1$ rounds.*

The proof of this proposition can be found in [94].

Proposition 6.10.9 *The Deterministic Combination Algorithm stops with success if and only if the input problem Γ has a solution in the free amalgamated product $\mathfrak{A}^\Sigma \otimes \mathfrak{B}^\Delta$.*

Proof. The subsequent proof is a variant of the proof for Proposition 4.16 of [94]. We present it here to gain a useful corollary.

First assume that the $\Sigma \cup \Delta$-constraint problem Γ has a solution. By Proposition 6.10.2 there exists a complete decision set C_L with $\Gamma_{\neq} \subset C_L$ such that the Σ-constraint problem $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_L)$ and the Δ-constraint problem $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$ are solvable. Let $C_1, \ldots, C_k$ $(k \geq 1)$ denote the sequence of universal cover points that are computed in the rounds of the Deterministic Combination Algorithm. Obviously, $C_1 = C_\bot \subseteq C_L$. Assume that $i < k$ and $C_i \subseteq C_L$. Let (Γ_I, C_i) be the input for round i, where $I \in \{\Sigma, \Delta\}$. The fact that C_{i+1} is a *universal* cover point for (Γ_I, C_i) implies that $C_{i+1} \subseteq C_L$. It follows that $C_k \subseteq C_L$, and the algorithm does not stop with failure. Hence it stops with success.

Now assume that the algorithm stops with success, say in round $l \geq 2$. Suppose that the input problem of round l is the I-constraint problem (Γ_I, C_l), where $I \in \{\Sigma, \Delta\}$. Choose an arbitrary *faithful* extension of C_l to a complete decision set C_L. Lemma 6.10.5 shows that such a faithful extension exists. Condition (ii) for round l ensures that C_L is also a faithful extension of C_{l-1}. Since C_l and C_{l-1} are cover points, we know that both (Γ_I, C_L) and (Γ_J, C_L) (with $\{I, J\} = \{\Sigma, \Delta\}$) are solvable constraint problems with decision set C_L. It follows from Proposition 6.10.2 that the $\Sigma \cup \Delta$-constraint problem Γ is solvable. ■

Corollary 6.10.10 *For every complete decision set C_L such that (Γ_Σ, C_L) and (Γ_Δ, C_L) are solvable, the last universal cover point C_k of the Deterministic Combination Algorithm lies below C_L, i.e., $C_k \subseteq C_L$.*

The following technical lemma on regular structures will be used frequently in subsequent proofs.

Lemma 6.10.11 *Let $(\mathfrak{A}^\Sigma, X)$ be a unitary regular non-collapsing quasi-free structure. Let (Γ, C) be constraint problem with decision set, where $\{u \dot{\mapsto} \Delta, v \dot{\mapsto} \Sigma, u \dot{<} v\} \subseteq C$. And let μ be the most general solution of (Γ, C) and σ another solution of it. If $\mu(u) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(v))$ then $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\sigma(v))$.*

Proof. Since μ is the most general solution and σ is a solution, there exists an endomorphism ν such that for all $x \in \mathit{Var}(\Gamma) : \sigma(x) = \nu\mu(x)$. Because $u \dot{\mapsto} \Delta \in C$ and σ is a solution, $\sigma(u) \in X$. From $\mu(u) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(v))$ follows $\nu\mu(u) \in \nu(\mathrm{Stab}^{\mathfrak{A}}(\mu(v)))$ which implies $\sigma(u) \in \nu(\mathrm{Stab}^{\mathfrak{A}}(\mu(v)))$. Therefore $\mathrm{Stab}^{\mathfrak{A}}(\sigma(u)) \subseteq \mathrm{Stab}^{\mathfrak{A}}(\nu(\mathrm{Stab}^{\mathfrak{A}}(\mu(v))))$. Since $\sigma(u) \in X$, we know $\mathrm{Stab}^{\mathfrak{A}}(\sigma(u)) = \sigma(u)$. Hence $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\nu(\mathrm{Stab}^{\mathfrak{A}}(\mu(v))))$, which implies $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\nu\mu(v))$ because ν is regular. Therefore $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\sigma(v))$. ■

Proposition 6.10.12 *Let $(\mathfrak{A}^\Sigma, X)$ be a unitary regular non-collapsing quasi-free structure. Then $(\mathfrak{A}^\Sigma, X)$ is $\mathcal{L}$-convex.*

Proof. Let (Γ, C_1) be a constraint problem. We have to show that there exists a universal cover point for (Γ, C_1).[4]

In the first case, there exists no complete decision set $C_L \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ such that $C_1 \subseteq C_L$ and (Γ, C_L) is solvable. In this case, $C_\top$ is a universal cover point for (Γ, C_1).

In the non trivial case, we assume there exists a complete decision set C_L such that $C_1 \subseteq C_L$ and (Γ, C_L) has solution σ. By definition, σ solves $\Gamma_1 := \Gamma \wedge \bigwedge_{u \doteq v \in C_1} u \doteq v$. Let μ be the most general solution of Γ_1, let

$$
\begin{array}{lcl}
C_= & := & \{u \doteq v \mid u, v \in \mathcal{U}, \mu(u) = \mu(v)\} \\
C_\Sigma & := & \{u \dot{\mapsto} \Sigma \mid u \in \mathcal{U}, \mu(u) \notin X\} \\
C_< & := & \{u \dot{<} v \mid u, v \in \mathcal{U}, \mu(u) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(v)), \mu(v) \notin X, u \dot{\mapsto} \Delta \in C_1\}
\end{array}
$$

and let C denote the closure of $C_1 \cup C_= \cup C_\Sigma \cup C_<$. Obviously, all equations of C are in $C_=$.

[4] Please note that X denotes the atom set, and $\mathcal{U}$ denotes the set of variables.

We have to show that C is a cover point for (Γ, C_1). Let C'_L be a complete decision set on $\mathcal{U}$ that faithfully extends C. We show that μ is a solution of (Γ, C'_L): Clearly, μ solves Γ; and since C'_L faithfully extends C also $\mu(u) = \mu(v)$ iff $u \doteq v \in C'_L$. For labelling decisions, let $u \in \mathcal{U}$ and $u \dot{\mapsto} \Delta \in C'_L$. Then $u \dot{\mapsto} \Sigma \notin C_\Sigma \subseteq C'_L$ and $\mu(u) \in X$. For ordering decisions, assume that $u \dot{\mapsto} \Sigma, v \dot{\mapsto} \Delta, u \dot{<} v \in C'_L$. Then $v \dot{\mapsto} \Sigma \notin C \subseteq C'_L$ and $\mu(v) \in X$. $u \dot{\neq} v \in C'_L$ because C'_L is closed, and $u \dot{\neq} v \in C$ because C'_L faithfully extends C. Hence $\mu(u) \neq \mu(v)$. Now suppose $\mu(v) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(u))$. Then $\mu(u) \notin X$ and $v \dot{<} u \in C \subseteq C'_L$, which is a contradiction since $u \dot{<} v \in C'_L$.

Remains to prove that C is a *universal* cover point for (Γ, C_1). Let C_L, σ, Γ_1 and μ as above. Since σ is a solution of Γ_1, there exists an endomorphism $\nu \in \mathit{End}_{\mathfrak{A}}^{\Sigma}$ such that $\sigma(v) = \nu\mu(v)$ for all $v \in \mathcal{U}$. In order to show that $C \subseteq C_L$ it suffices to prove that $(C_= \cup C_\Sigma \cup C_\Longrightarrow) \subseteq C_L$ since C_L is closed and $C_1 \subseteq C_L$.

Firstly, let $u \doteq v \in C_=$. Then $\mu(u) = \mu(v)$ and $\sigma(u) = \nu\mu(u) = \nu\mu(v) = \sigma(v)$. By definition, $u \doteq v \in C_L$.

Secondly, let $u \dot{\mapsto} \Sigma \in C_\Sigma$. Then $\mu(u) \notin X$, the set of atoms. Since ν is non-collapsing, also $\sigma(u) = \nu\mu(u) \notin X$. Hence $u \dot{\mapsto} \Delta \notin C_L$ by definition of a solution, and thus $u \dot{\mapsto} \Sigma \in C_L$ since C_L is a complete decision set.

Thirdly, let $u \dot{<} v \in C_<$. Then $u \dot{\mapsto} \Delta \in C_1 \subseteq C_L$. From the definition of $C_<$ it follows that $\mu(u) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(v))$ and $\mu(v) \notin X$. Then $\sigma(v) = \nu\mu(v) \notin X$, because ν is non-collapsing. Hence $v \dot{\mapsto} \Delta \notin C_L$ and thus $v \dot{\mapsto} \Sigma \in C_L$. Since $u \dot{\mapsto} \Delta \in C_L$ we know $\sigma(u) \in X$ by definition of a solution. Finally, $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\sigma(v))$ by Lemma 6.10.11. Because u and v have different labels in C_L, they must be ordered due to closure of C_L. Thus either $u \dot{<} v \in C_L$ or $v \dot{<} u \in C_L$. But σ is a solution and $\sigma(u) \in \mathrm{Stab}^{\mathfrak{A}}(\sigma(v))$, therefore $u \dot{<} v \in C_L$. ■

6.10.2 Deterministic Combination of Quasi-free Structures is Unitary

Thus the task is to show that deterministic combination of $\mathcal{L}$-convex unitary quasi-free structures is unitary. To do this, we prove that a regular non-collapsing quasi-free structure is unitary with respect to solving constraint problems with decision sets provided it is unitary for problems with constants. And we have to show that every solution of a constraint problem in the free amalgamated product is an instance of the solution that the Deterministic Combination Algorithm computes.

Lemma 6.10.13 *Let $(\mathfrak{A}^\Sigma, X)$ be a non-collapsing regular quasi-free structure. If $(\mathfrak{A}^\Sigma, X)$ is unitary with respect to constraint problems with constants, then it is unitary with respect to problems with decision sets.*

Proof. Let (Γ, C) be a problem with decision set $C \in \mathcal{C}_{\mathcal{L}(Y)}$. The equality decisions $C_=$ give rise to an equivalence relation on the set of variables Y.

Choose a representant for each equivalence class and replace each variable in (Γ, C) by its representant to obtain (Γ_1, C_1). Clearly, (Γ, C) and (Γ_1, C_1) are equisolvable[5]. Now for each labelling decision $u \dot{\mapsto} \Delta \in C_1$ with $\Delta \neq \Sigma$ choose a *new* constant $\bar{u}$ (not contained in Σ or Γ) and replace u by $\bar{u}$ in (Γ_1, C_1) obtaining (Γ_2, C_2). Again, (Γ_1, C_1) and (Γ_2, C_2) are equisolvable, and the equality and labelling decisions of C are now coded into Γ_2. Let μ be the most general solution of Γ_2, and let $\bar{u} \dot{<} x \in C_2$. If $\mu(\bar{u}) \in \mathrm{Stab}^{\mathfrak{A}}(\mu(x))$ then for every instance σ of μ also $\sigma(\bar{u}) \in \mathrm{Stab}^{\mathfrak{A}}(\sigma(x))$ by Lemma 6.10.11 since $\bar{u} \notin \Sigma$. Therefore, if μ violates an ordering decision, then so does every instance of it. So, let σ be a solution of (Γ_2, C_2). Then σ is an instance of μ and the above shows that μ solves (Γ_2, C_2). ∎

Before we can proof the main theorem, we need a few technical lemmata.

Lemma 6.10.14 *Let $(\mathfrak{A}^{\Sigma}, X)$ be a unitary non-collapsing regular quasi-free structure and Γ an elementary constraint problem. Let $C_1, C_2 \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ and $C_1 \subseteq C_2$. Let μ be the most general solution of (Γ, C_1) and σ a solution of (Γ, C_2). Then σ is an instance of μ.*

Proof. Since $C_1 \subseteq C_2$, clearly σ also solves (Γ, C_1). If μ is the most general solution of (Γ, C_1), then σ is an instance of it. ∎

Lemma 6.10.15 *Let $(\mathfrak{A}^{\Sigma}, X)$ be a unitary non-collapsing regular quasi-free structure and let Γ be an elementary constraint problem. Let $C \in \mathcal{C}_{\mathcal{L}(\mathcal{U})}$ and μ the most general solution of (Γ, C). Let $C_L \supseteq C$ be a* faithful *extension to a complete decision set such that (Γ, C_L) is solvable. Then μ is the most general solution of (Γ, C_L).*

Proof. Let σ be a solution of (Γ, C_L). Then σ is an instance of μ by the above lemma, i.e., there is an endomorphism $\nu \in \mathcal{M}$ with $\sigma = \nu\mu$. Therefore if μ is a solution, it is the most general one. So, we show that μ is a solution of (Γ, C_L). The only decisions of C_L that concern us are ones not contained in C. Since C_L is a faithful extension, these can only be labelling or ordering decisions. Let $x \dot{\mapsto} \Delta \in C_L \setminus C$ where $\Delta \neq \Sigma$. Then $\sigma(x) \in X$ and also $\mu(x) \in X$, because $(\mathfrak{A}^{\Sigma}, X)$ is non-collapsing. Let $u \dot{\mapsto} \Delta, x \dot{\mapsto} \Sigma, u \dot{<} x \in C_L$. Then $\sigma(u) \notin \mathrm{Stab}^{\mathfrak{A}}(\sigma(x))$ because σ is a solution. Hence $\mu(u) \notin \mathrm{Stab}^{\mathfrak{A}}(\mu(x))$ due to Lemma 6.10.11. Thus μ is a solution of (Γ, C_L). ∎

Lemma 6.10.16 *Let $\sigma \geq \mu$ and a_1, a_2 be two qf-isomorphisms. Then $a_1\sigma \geq a_2\mu$.*

Proof. If $\sigma \geq \mu$, then there is an endomorphism ν such that $\sigma = \nu\mu$. Now $a_1\sigma = a_1\nu\mu = a_1\nu\mathrm{id}\mu = a_1\nu a_2^{-1}a_2\mu$. Hence $a_1\nu a_2^{-1}$ is an endomorphism ν' such that $a_1\sigma = \nu' a_2\mu$, which shows $a_1\sigma \geq a_2\mu$. ∎

[5]There is an NP-algorithm that translates solutions of one into solutions of the other and back.

Theorem 6.10.17 *Let Γ be a $\Sigma \cup \Delta$-constraint problem of $\mathfrak{A}^\Sigma \otimes \mathfrak{B}^\Delta$ and σ a solution of Γ. Then σ is an instance of μ, the solution that the Deterministic Combination Algorithm computes.*

Proof. We assume Γ is in decomposed form $\Gamma_\Sigma \wedge \Gamma_\Delta \wedge \Gamma_{\neq}$. Since Γ has solution, the Deterministic Combination Algorithm stops with success by Proposition 6.10.9, say after k steps with universal cover point C_k. By Lemma 6.10.13, there exists a most general solution μ_Σ of the constraint problem $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_k)$ and a most general solution μ_Δ of $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_k)$. Choose an arbitrary faithful extension to a complete decision set C_K of C_k. By Lemma 6.10.15, μ_Σ is the most general solution of $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_K)$ and μ_Δ is the most general solution of $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_K)$. Thus $\mu := \mu_\Sigma \otimes \mu_\Delta$ is a solution of Γ by Corollary 6.10.3.

Now, let σ be a solution of Γ. There exists a complete decision set C_L and solutions σ_Σ of $(\Gamma_\Sigma \wedge \Gamma_{\neq}, C_L)$ and σ_Δ of $(\Gamma_\Delta \wedge \Gamma_{\neq}, C_L)$ with $\sigma = \sigma_\Sigma \otimes \sigma_\Delta$ by Proposition 6.10.2. By Corollary 6.10.10, $C_k \subseteq C_L$. Therefore σ_Σ is an instance of μ_Σ and σ_Δ is an instance of μ_Δ by Lemma 6.10.14. By Corollaries 6.10.3 and 6.2.12, the combined solutions $\sigma = \sigma_\Sigma \otimes \sigma_\Delta$ and $\mu = \mu_\Sigma \otimes \mu_\Delta$ are constructed out of the component solutions solely by isomorphisms. Hence σ is an instance of μ by Lemma 6.10.16. ∎

6.11 Conclusion

In this chapter, we analysed the role of negation in the combination of constraint systems. In the first part, we were concerned with how to reduce solvability of mixed constraints in the free amalgamated product of two quasi-free structures to solvability of pure constraints in the components. We showed that the existential theory of the free amalgamated product of two quasi-free structures is decidable, if the solvability of pure constraint problems with linear constant restrictions is decidable in both components. Furthermore, the existential theory of the ground substructure of the free amalgamated product is decidable, provided the existence of restrictive solutions for pure constraint problems with linear constant restrictions is decidable in both components. We saw that we cannot solve a large quantifier fragment in the free amalgamated product than the existential theory and that the technical notion of a linear constant restriction could not be translated into purely logical terms. We also had to learn that rational amalgamation is not suitable for combining constraint systems with negation in the constraint languages.

In the second part, we discussed the independence of negative constraints property. We started with a look at the independence property in the context of equational unification to find that unitary equational theories have the independence property, while finitary have not. We also presented a modularity result for equational unification stating that the union of two signature-disjoint equational theories is unitary, if the components are unitary, regular and collapse-free.

We lifted these results to quasi-free structures. A quasi-free structure has the independence property, if it is unitary. And the final modularity result states that the free amalgamation of unitary regular and non-collapsing quasi-free structures is again unitary and thus has the independence property.

List of Theorems

Bibliography

[1] Peter Aczel. *Non-wellfounded Sets.* Number 14 in CSLI Lecture Notes. CSLI, Stanford University, USA, 1988.

[2] Hassan Ait-Kaci, Andreas Podelski, and Gert Smolka. A Feature-based Constraint System for Logic Programming With Entailment. *Theoretical Computer Science*, 122:263–283, 1994.

[3] Andre Arnold and Maurice Nivat. The Metric Space of Infinite Trees. Algebraic and Topological Properties. *Fundamenta Informaticae*, 3(4):445–476, 1980.

[4] Franz Baader. Unification Properties of Commutative Theories: A Categorical Treatment. In *Proceedings of the Conference on Category Theory and Computer Science*, LNCS 389, pages 273–299, Manchester (UK), 1989. Springer–Verlag.

[5] Franz Baader and Klaus Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. In Deepak Kapur, editor, *Automated Deduction, Proceedings CADE-11*, LNAI 607, pages 50–65. Springer-Verlag, 1992.

[6] Franz Baader and Klaus Schulz. Combination Techniques and Decision Problems for Disjunification. In Claude Kirchner, editor, *Rewriting Techniques and Applications, Proceedings RTA-93*, LNCS 690, pages 301–315. Springer-Verlag, 1993.

[7] Franz Baader and Klaus U. Schulz. Combination of Constraint Solving Techniques: An Algebraic Point of View. Technical Report CIS-Bericht-94-75, CIS, Universität München, 1994. Long version of [8].

[8] Franz Baader and Klaus U. Schulz. Combination of Constraint Solving Techniques: An Algebraic Point of View. In Jieh Hsiang, editor, *Rewriting Techniques and Applications, Proceedings RTA-95*, LNCS 914, pages 352–366. Springer-Verlag, 1995. Short version of [7].

[9] Franz Baader and Klaus U. Schulz. Combination Techniques and Decision Problems for Disunification. *Theoretical Computer Science*, 142:229–255, 1995.

[10] Franz Baader and Klaus U. Schulz. On the Combination of Symbolic Constraints, Solution Domains, and Constraint Solvers. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming, Proccedings CP'95*, LNCS 976, pages 380–397. Springer-Verlag, 1995.

[11] Franz Baader and Klaus U. Schulz. On the Combination of Symbolic Constraints, Solution Domains, and Constraint Solvers. Technical Report CIS-Bericht-95-120, CIS, Universität München, 1995.

[12] Franz Baader and Klaus U. Schulz. Combination of Constraint Solvers for Free and Quasi-Free Structures. Technical Report CIS-Bericht-96-90, CIS, Universität München, 1996.

[13] Franz Baader and Klaus U. Schulz, editors. *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, volume 3 of *Applied Logic Series*. Kluwer Academic Publishers, 1996.

[14] Franz Baader and Klaus U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation*, 21:211–243, 1996.

[15] Franz Baader and Klaus U. Schulz. Combination of Constraint Solvers for Free and Quasi-Free Structures. *Theoretical Computer Science*, 192:107–161, 1998.

[16] Franz Baader and Klaus U. Schulz. Unification Theory. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, pages 225–263. Kluwer Academic Publishers, 1998.

[17] Franz Baader and Jörg H. Siekmann. Unification Theory. In Dov M. Gabbay, Christopher J. Hogger, and John Alan Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 41–125. Oxford University Press, 1994.

[18] Leo Bachmair. *Canonical equational proofs.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[19] Rolf Backofen and Gert Smolka. A Complete and Recursive Feature Theory. *Theoretical Computer Science*, 146:243–268, 1995.

[20] Alexandre Boudet. Unification in a Combination of Equational Theories: An Efficient Algorithm. In *Stickel* [109], pages 292–307, 1990.

[21] Alexandre Boudet. Combining Unification Algorithms. *Journal of Symbolic Computation*, 16(6):597–626, 1993.

[22] Alexandre Boudet, Jean-Pierre Jouannaud, and Manfred Schmidt-Schauß. Unification in Boolean Rings and Abelian Groups. *Journal of Symbolic Computation*, 8:449–477, 1989.

[23] Alan Bundy, editor. *Automated Deduction, Proceedings CADE-12, Nancy, France*, LNAI 814. Springer-Verlag, 1994.

[24] Hans-Jürgen Bürckert. A Resolution Principle for Clauses with Constraints. In *Stickel* [109], pages 178–192, 1990.

[25] Ta Chen and Siva Anantharaman. STORM: A Many-to-one Associative-commutative Matcher. In Jieh Hsiang, editor, *Rewriting Techniques and Applications, Proceedings RTA-95*, LNCS 914, pages 414–419. Springer-Verlag, 1995.

[26] Greg Cherlin. *Model Theoretic Algebra: Selected Topics.* Number 521 in Lecture Notes in Mathematics. Springer-Verlag, 1976.

[27] Noam Chomsky. *Lectures on Government and Binding.* Foris Publications, Dordrecht, Holland, 1981.

[28] Paul M. Cohn. *Universal Algebra.* Harper & Row, New York, 1965.

[29] Alain Colmerauer. Equations and Inequations on Finite and Infinite Trees. In Institute for New Generation Computer Technology, editor, *Proceedings of the 2nd International Conference on Fifth Generation Computing Systems*, pages 85–99, Tokyo, 1984. Ohmsha et al.

[30] Alain Colmerauer. An Introduction to PROLOG III. *Communications of the ACM*, 33:69–90, 1990.

[31] Hubert Comon. *Unification et disunification: Théorie et applications.* PhD thesis, I.N.P. de Grenoble, France, 1988.

[32] Hubert Comon. Disunification: A Survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic*, pages 322–359. MIT Press, 1991.

[33] Hubert Comon and Pierre Lescanne. Equational Problems and Disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[34] Tom Cornell. *Description Theory, Licensing Theory and Principle-Based Grammars and Parsers.* PhD thesis, University of California at Los Angeles, 1992.

[35] Bruno Courcelle. Infinite Trees in Normal Form and Recursive Equations Having a Unique Solution. *Mathematical Systems Theory*, 13:131–180, 1979.

[36] Bruno Courcelle. Fundamental Properties of Infinite Trees. *Theoretical Computer Science*, 25:95–169, 1983.

[37] Nachum Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[38] Eric Domenjoud. A Technical Note on AC-Unification. The Number of Minimal Unifiers of the Equation $\alpha x_1 + \cdots + \alpha x_p \doteq_{AC} \beta y_1 + \cdots + \beta y_q$. *Journal of Automated Reasoning*, 8:39–44, 1992.

[39] Jochen Dörre and Andreas Eisele. Unification of Disjunctive Feature Descriptions. In *Proceedings of the 26th Annual Meeting of the ACL, State University of New York at Buffalo*, pages 286–294, Buffalo, New York, 1988.

[40] Jochen Dörre and Andreas Eisele. Feature Logic with Disjunctive Unification. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 2, pages 100–105, Helsinki, Finland, 1990.

[41] Manfred Droste and Rüdiger Göbel. Universal Domains and the Amalgamation Property. *Mathematical Structures in Computer Science*, 3:137–159, 1993.

[42] François Fages. Associative-commutative Unification. In Robert E. Shostak, editor, *7th International Conference on Automated Deduction, Proceedings CADE-7*, LNCS 170, pages 194–208, New York, 1984. Springer-Verlag.

[43] François Fages. Associative-Commutative Unification. *Journal of Symbolic Computation*, 3:257–275, 1987.

[44] Dov M. Gabbay. An Overview of Fibred Semantics and the Weaving of Logics. In Franz Baader and Klaus U. Schulz, editors, *Frontiers of Combining Systems*, volume 3 of *Applied Logic Series*, pages 1–55. Kluwer Academic Publishers, 1996.

[45] Dale Gerdemann. Parsing as Tree Traversal. In *COLING 94, Proceedings of the 15 International Conference on Computational Linguistics*, pages 396–400, 1994.

[46] Dale Gerdemann and Paul King. The Correct and Efficient Implementation of Appropriateness Specifications for Typed Feature Structures. In *COLING 94, Proceedings of the 15 International Conference on Computational Linguistics*, pages 956–960, 1994.

[47] George Grätzer. *Universal Algebra.* Springer-Verlag, Berlin, second edition, 1979.

[48] Claudio Gutiérrez. Satisfiability of Word Equations with Constants is in Exponential Space. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science, FOCS 98*, pages 112–119. IEEE Computer Science Press, 1998.

[49] Martin Henz, Gert Smolka, and Jörg Würtz. Oz-A Programming Language for Multi-Agent Systems. In Ruzena Bajcsy, editor, *13th International Joint Conference on Artificial Intelligence*, volume 1, pages 404–409. Morgan Kaufmann Publishers, 1993.

[50] Jacques Herbrand. Recherches sur la Théorie de la Démonstration. *Travaux de la Societé des Sciences et des Lettres de Varsovie, Classe III*, 33(128), 1930. Reprinted in [51].

[51] Jacques Herbrand. *Logical Writings*. Edited by Warren D. Goldfarb. D. Reidel Publishing Company, Dordrecht, Holland, 1971.

[52] Alexander Herold. Combination of Unification Algorithms. In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Proceedings CADE-8*, LNCS 230, pages 450–469. Springer-Verlag, 1986.

[53] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg H. Siekmann. KEIM: A Toolkit for Automated Deduction. In *Bundy* [23], pages 807–810, 1994.

[54] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[55] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19:503–581, 1994.

[56] Mark Johnson. Deductive Parsing: The Use of Knowledge of Language. In *Principle-Based Parsing: Computation and Psycholinguistics*, pages 39–64. Kluwer, Dordrecht, 1991.

[57] Mark Johnson. Constraint-based Natural Language Parsing. Ms., Brown University/ESSLLI 1995 coursenotes, August 1995.

[58] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a Set of Rules modulo a set of Equations. *SIAM Journal on Computing*, 15:1155–1195, 1986.

[59] Deepak Kapur and Paliath Narendran. Complexity of Unification Problems with Associative-Commutative Operators. *Journal of Automated Reasoning*, 9:261–288, 1992.

[60] Stephan Kepser. Negation in Combining Constraint Systems. In Maarten de Rijke and Dov M. Gabbay, editors, *Frontiers of Combining Systems'98*, APLS. Kluwer Academic Publishers, 1998.

[61] Stephan Kepser and Jörn Richts. Optimisation Techniques for Combining Unification Algorithms. Technical Report LTCS-Report-96-04, Theoretical Computer Science Lab, RWTH Aachen, Germany, 1996. Available at `ftp://www-lti.informatik.rwth-aachen.de/pub/papers/1996/LTCS-Rep-96-04.ps.gz`.

[62] Stephan Kepser and Jörn Richts. Optimisation Techniques for Combining Constraint Solvers. In Maarten de Rijke and Dov M. Gabbay, editors, *Frontiers of Combining Systems'98*, APLS. Kluwer Academic Publishers, 1998.

[63] Stephan Kepser and Klaus U. Schulz. Combination of Constraint Systems II: Rational Amalgamation. In Eugene Freuder, editor, *Principles and Practice of Constraint Programming, Proceedings CP96*, LNCS 1118, pages 282–296. Springer–Verlag, 1996. Long version in [97].

[64] Claude Kirchner. Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Equationelles. Thèse d'État, Université de Nancy I, France, 1985.

[65] Claude Kirchner. From Unification in Combination of Equational Theories to a New AC-unification Algorithm. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 171–210. Academic Press, New York, 1987.

[66] Claude Kirchner and Hélène Kirchner. Constrained Equational Reasoning. In Gaston H. Gonnet, editor, *Proceedings of SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation: ISSAC'89*, pages 382–389. ACM Press, 1989.

[67] Mike Livesey and Jörg H. Siekmann. Unification of *AC*-terms (Bags) and *ACI*-terms (Sets). Technical Report 3-76, Institut für Informatik I, University of Karlsruhe, 1976.

[68] Roger C. Lyndon. Properties Preserved Under Homomorphisms. *Pacific Journal of Mathematics*, 9:143–154, 1959.

[69] Benjamín Macías. *An Incremental Parser for Government-Binding Theory.* PhD thesis, University of Cambridge, 1990.

[70] Michael J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *3rd Logic in Computer Science Conference.* IEEE, 1988.

[71] Gennadij Semjonovitch Makanin. The Problem of Solvability of Equations in a Free Semigroup. *Mat. USSR Sbornik*, 32, 1977.

[72] Anatolij Ivanovič Mal'cev. *The Metamathematics of Algebraic Systems.* Edited by Benjamin Franklin Wells, volume 66 of *Studies in Logic.* North-Holland Publishing Company, 1971.

[73] Anatolij Ivanovič Mal'cev. *Algebraic Systems.* Volume 192 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellung.* Springer-Verlag, Berlin, 1973.

[74] Karl Meinke and John V. Tucker. Universal Algebra. In Samson Abramski, Dov M. Gabbay, and Tom S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 189 – 411. Clarendon Press, Oxford, UK, 1992.

[75] M. Andrew Moshier and Carl J. Pollard. The Domain of Set-valued Feature Structures. *Linguistics and Philosophy*, 17:607–631, 1994.

[76] Greg Nelson. Combining Satisfiability Procedures by Equality-sharing. *Contemporary Mathematics*, 29:201–211, 1984.

[77] Greg Nelson and Derek C. Oppen. Simplification by Cooperation of Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[78] Robert Nieuwenhuis and Albert Rubio. *AC*-superposition with Constraints: No *AC*-unifier Needed. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction, Nancy, France*, LNAI, pages 545–559. Springer, 1994.

[79] Werner Nutt. Unification in Monoidal Theories. In *Stickel* [109], pages 618–632, 1990.

[80] Derek C. Oppen. Complexity, Convexity and Combination of Theories. *Theoretical Computer Science*, 12:291–302, 1980.

[81] Jochen Pfalzgraf. Logical Fiberings and Polycontextural Systems. In Philippe Jorrand and Jozef Kelemen, editors, *Fundamentals of Artificial Intelligence Research*, LNAI 535, pages 170–184. Springer-Verlag, 1991.

[82] Jochen Pfalzgraf and Karel Stokkermans. On Robotics Scenarios and Modeling with Fibred Structures. In Jochen Pfalzgraf and Dongming Wang, editors, *Automated Practical Reasoning: Algebraic Approaches*, Springer Series Texts and Monographs in Symbolic Computation, pages 53–80. Springer-Verlag, 1995.

[83] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, Cambridge, USA, 1991.

[84] Don Pigozzi. The Join of Equational Theories. *Colloquium Mathematicum*, XXX:15–25, 1974.

[85] Gordon D. Plotkin. Building-in Equational Theories. *Machine Intelligence*, 7:73–90, 1972.

[86] Carl J. Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Volume I: Fundamentals.* Number 13 in CSLI Lecture Notes. CSLI, Stanford University, USA, 1987.

[87] Carl J. Pollard and Ivan A. Sag. *Head-driven Phrase Structure Grammar.* Studies in Contemporary Linguistics. CSLI & University of Chicago Press, 1994.

[88] Christophe Ringeissen. Cooperation of Decision Procedures for the Satisfiability Problem. In *Baader & Schulz* [13], pages 121–140, 1996.

[89] John Alan Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[90] William C. Rounds. Set Values for Unification Based Grammar Formalisms and Logic Programming. Technical Report CSLI-88-129, CSLI, Stanford University, 1988.

[91] Albert Rubio. Theorem Proving modulo Associativity. In Hans Kleine-Büning, editor, *Computer Science Logic, Proceedings CSL'95, Paderborn, Germany*, LNCS 1092, pages 452–467. Springer, 1995.

[92] Manfred Schmidt-Schauß. Unification in a Combination of Arbitrary Disjoint Equational Theories. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction, Proceedings CADE-9*, LNCS 310, pages 378–396. Springer-Verlag, 1988.

[93] Manfred Schmidt-Schauß. Unification in a Combination of Arbitrary Disjoint Equational Theories. *Journal of Symbolic Computation*, 8(1,2):51–99, 1989.

[94] Klaus U. Schulz. Combining Unification and Disunification Algorithms—Tractable and Intractable Instances. Technical Report CIS-Bericht-96-99, CIS, Universität München, 1996.

[95] Klaus U. Schulz. A Criterion for Intractability of E-unification with Free Function Symbols and Its Relevance for Combination of Unification Algorithms. In Hubert Comon, editor, *Rewriting Techniques and Applications, Proceedings RTA-97*, LNCS 1232, pages 284–298. Springer-Verlag, 1997.

[96] Klaus U. Schulz. Tractable and Intractable Instances of Combination Problems for Unification and Disunification. *Journal of Logic and Computation*, 1999.

[97] Klaus U. Schulz and Stephan Kepser. Combination of Constraint Systems II: Rational Amalgamation. Technical Report CIS-Bericht-96-86, CIS, Universität München, 1996. Long version of [63].

[98] Klaus U. Schulz and Stephan Kepser. Combination of Constraint Systems II: Rational Amalgamation. *Theoretical Computer Science*, 1999.

[99] Robert E. Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

[100] Robert E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1):1–12, 1984.

[101] Thoralf Skolem. Über einige Satzfunktionen in der Arithmetik. *Skrifter, Vitenskapsakademiet i Oslo*, I(7):1–28, 1930. Reprinted in [102].

[102] Thoralf Skolem. *Selected Works in Logic.* Edited by Jens Erik Fenstad. Universitetsforlaget, Oslo, 1970.

[103] Gert Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, LNCS 1000, pages 324–343, Berlin, 1995. Springer-Verlag.

[104] Gert Smolka and Ralf Treinen. Records for Logic Programming. *Journal for Logic Programming*, 18(3):229–258, 1994.

[105] Dennis Stanton and Dennis White. *Constructive Combinatorics*. Undergraduate Texts in Mathematics. Springer Verlag, Berlin, New York, 1986.

[106] Mark E. Stickel. A Complete Unification Algorithm for Associative-commutative Functions. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 71–82, Tblisi, USSR, 1975.

[107] Mark E. Stickel. A Unification Algorithm for Associative-Commutative Functions. *Journal of the ACM*, 28(3):423–434, 1981.

[108] Mark E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Deduction*, 1(4):333–356, 1985.

[109] Mark E. Stickel, editor. *Automated Deduction, Proceedings CADE-10*, LNAI 449, Berlin, Germany, 1990. Springer–Verlag.

[110] Ian Sutherland. *A Man-machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Techology, 1963.

[111] Erik Tidén. Unification in Combinations of Collapse-free Theories with Disjoint Sets of Function Symbols. In Jörg H. Siekmann, editor, *International Conference on Automated Deduction, Proceedings CADE-8*, LNCS 230, pages 431–449. Springer-Verlag, 1986.

[112] Cesare Tinelli and Mehdi Harandi. A New Correctness Proof of the Nelson–Oppen Combination Procedure. In *Baader & Schulz* [13], pages 103–120, 1996.

[113] Ralf Treinen. A New Method for Undecidability Proofs of First Order Theories. *Journal of Symbolic Computation*, 14:437–457, 1992.

[114] Peter Van Roy, Michael Mehl, and Ralf Scheidhauer. Integrating efficient records into concurrent constraint programming. In *8th International Symposium on Programming Languages, Implementations, Logic, and Programs (PLILP96), Aachen*, September 1996.

[115] Hugo Volger. Principle Languages and Principle Based Parsing. Technical Report Arbeitspapiere des Sonderforschungsbereichs 340; 82, Universität Stuttgart, 1997.

[116] Nikolai Weaver. Generalized Varieties. *Algebra Universalis*, 30:27–52, 1993.

[117] Katherine A. Yelick. Combining Unification Algorithms for Confined Regular Equational Theories. In Jean-Pierre Jouannaud, editor, *Rewriting Techniques and Applications, Proceedings RTA-85*, LNCS 202, pages 365–380. Springer-Verlag, 1985.

[118] Katherine A. Yelick. Unification in Combinations of Collapse-Free Regular Theories. *Journal of Symbolic Computation*, 3:153–181, 1987.