STELLINGEN

behorende bij het proefschrift

_A Theory of Many-sorted Evolving Algebras_

Hans Tonino

1. In Pure Type Systemen waarin de typetoekenning aan termen uniek is, geldt de eigenschap van _subject conversie_. Dit wil zeggen, als \( \Gamma \vdash A : C \) en \( A =_\beta B \), dan \( \Gamma \vdash B : C \).


2. De notatie voor het resultaat van een substitutie op functiesymbolen met behulp van _extended function symbols_ zoals geïntroduceerd in dit proefschrift, is veel inzichtelijker dan de gebruikelijke notatie met behulp van de constructie _if . . . then . . . else . . ._.

(Zie: Hoofdstuk 3 van dit proefschrift.)

3. In de _E_-logica geldt voor elke term \( t \) dat \( \vdash t \neq \bot \). Dit zegt echter niets over de denotatie van een term \( t \); deze kan best ongedefinieerd zijn.

4. Het controleren van een wiskundig bewijs kan grotendeels worden opgevat als een _empirisch experiment_: men zal een bepaalde representatie van een bewijs moeten onderwerpen aan een welbepaalde procedure. Het inzetten van computers voor deze taak leidt tot meer inzicht in de structuur van bewijzen en kan worden gezien als een nieuwe vorm van _peer reviewing_.

5. Nu er te veel door de KNAW erkende onderzoekscholen bestaan, gaat men over tot de oprichting van _toponderzoekscholen en technologische topinstituten_. De gewone wetenschapper wordt steeds meer gevangen in een bestuurlijke toren van Babel.

7. Madame Bovary van Flaubert kan worden opgevat als evidentie voor de stelling dat het cultureel relativisme onjuist is.

8. Een goed wiskundeboek is als een goede uitvoering van een muziekstuk. De stellingen kunnen dan worden vergeleken met de thema's, en de bewijzen van de stellingen met de interpretatie van de thema's.


10. Het utiliteitsdenken waaraan veel beleidsmakers zich tegenwoordig schuldig maken, heeft als verborgen aannames dat het menselijk handelen voornamelijk moet worden bepaald door economisch nut.

11. Het zogenaamde ethisch reveal van sommige multi-nationals komt eerder voort uit winstooogmerk dan uit ethische motieven.

12. Om n managers bij elkaar te krijgen voor een vergadering, dient men in de orde $2^n$ mogelijke data voor te leggen aan ieder van hen.

13. De hoeveelheid leerwerk om de tafels van vermenigvuldiging van 1 tot en met 10 te memoriseren kan met 45% worden verminderd indien rekening wordt gehouden met het feit dat de vermenigvuldiging commutatief is.
A Theory of
Many-sorted Evolving Algebras
A Theory of
Many-sorted Evolving Algebras

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.dr.ir. J. Blaauwendaal,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College van Dekanen aangewezen,
op donderdag 19 juni 1997 te 13.30 uur

door

Joannes Franciscus Maria TONINO

wiskundig ingenieur
geboren te 's-Gravenhage
Dit proefschrift is goedgekeurd door de promotor:
Prof.dr. S.C. van Westrhenen

Toegevoegd promotor:
Dr.ir. R. Sommerhalder

Samenstelling promotiecommissie:
Rector Magnificus
Prof.dr. S.C. van Westrhenen
Dr.ir. R. Sommerhalder
Prof.dr. H.P. Barendregt
Prof. S.J. Doorman M.Sc.
Prof.dr. J.-J.Ch. Meyer
Dr. R.P. Nederpelt
Prof.dr.ir. H.J. Sips

Technische Universiteit Delft, voorzitter
Technische Universiteit Delft, promotor
Technische Universiteit Delft, toegevoegd promotors
Katholieke Universiteit Nijmegen
Technische Universiteit Delft
Universiteit Utrecht
Technische Universiteit Eindhoven
Technische Universiteit Delft
Voorwoord


Aan de leden van de promotiecommissie. Ik voel mij vereerd dat ik in de gelegenheid word gesteld hun vragen te mogen beantwoorden.

Aan mijn directe collega’s Job, Cees, Henk, André, David, Marcel en Leon. Dit zijn de mensen met wie ik elke dag lunch en die ik elke avond ‘welterusten’ wens, onze code voor ‘een prettige avond en tot morgen’.

Aan Coby, Marian en Toos. Zij hadden precies door, wanneer ik met een puzzel worstelde en probeerden mij dan met hun humor op andere gedachten te brengen. Door die oogkleppen lukte dit niet altijd, maar wel vaak.

Aan al mijn vrienden en vriendinnen. Het wordt tijd weer eens met hen door te zakken.

Aan Ben. Hij heeft de mooie kant ontworpen en getekend.
Aan Greetje en Moniek. Omdat zij mijnparanimfen willen zijn.
Aan Barend, die fliessend(e) foto’s gaat maken.

Aan mijn vader, moeder, schoonmoeder, Severijn, Machiel en alle andere familieleden. Zij vroegen steeds “Waar gaat het over?” of “Hoe ver ben je nu?”
In het eerste geval stond ik meestal met mijn mond vol tanden. In het tweede geval ook ...


Hans Tonino

Delft, april 1997
Voorwoord
## Contents

1 Introduction and summary .......................................................... 1

2 Introduction to Evolving Algebras ........................................... 7
   2.1 Why Evolving Algebras? ..................................................... 7
   2.2 What is an Evolving Algebra? ........................................... 11

3 The syntax of *E*-logic .......................................................... 17
   3.1 Signatures, terms, and formulae ....................................... 17
   3.2 Substitution ............................................................... 23
   3.3 A Fitch-style deduction system for *E*-logic .................... 27

4 The semantics of *E*-logic ...................................................... 41
   4.1 Partial structures and interpretations .............................. 41
   4.2 Operations on interpretations ......................................... 47
   4.3 Soundness and completeness .......................................... 63
   4.4 Standard *E*-logic ....................................................... 74

5 Syntax and semantics of Evolving Algebras ............................... 79
   5.1 Syntax of Evolving Algebras .......................................... 79
   5.2 Operational semantics of Evolving Algebras .................... 87
   5.3 Adequacy results ....................................................... 104

6 A proof system for Evolving Algebras .................................... 137
   6.1 The proof system ........................................................ 137
   6.2 Example of a correctness proof ...................................... 165

7 Conclusions and future work .................................................. 175

References ................................................................................. 181
Chapter 1

Introduction and summary

Nowadays, our society cannot be imagined without the production, the processing and the trade of information. Except for the last activity, these can be considered to be important areas of computer science. Since the first computers came into use, the way programmers think about information has changed. In the beginning, information consisted of scientific data: to have information meant to know the value of some physical quantity under certain circumstances. This time lies far behind us now.

As computers evolved, the notion of information evolved as well. In modern functional languages the value of a variable can be even a data type. For instance, modern languages like Haskell (for a gentle introduction, see [HF92]) offer the possibility of defining polymorphic types in which free variables are used to denote any type. Also notable are the type classes in functional and object oriented languages.

As the application areas in which computers were used, became more and more complex, the need for more advanced programming methods and the need for methods to structure information did grow. To overcome the first problem, computer scientists invented specification formalisms by which the input-output behaviour of programs could be specified in a structured way, using sound logical formalisms. The second problem was attacked by designing languages which offer problem oriented data type systems. Both specification languages and type systems were (and still are) studied by logicians.

Although formal methods for specifying the syntax of programming languages, like the Backus-Naur Formalism, already existed for a long time, the first specification languages in which the semantics of computer programs could be concisely expressed, did not see daylight until the end of the seventies. Well-known methods include the Vienna Development Method (see e.g. [Jon90]) and the Z-method (see e.g. [Dil90]). Besides these state based methods, also other kinds of methods were developed. An example of a so-called algebraic specification language is OBJ (see [GW88]), founded on equational logic. We also have to mention Coloured Petri Nets (see [Jen92]), which are special condition-event systems. Very sophisticated is the proof-as-program paradigm of Constructive Type Theory (see e.g. [NPS90]), which is based on the so-called
Chapter 1. Introduction and summary

Curry-Howard-De Bruijn isomorphism (see [Gen93, TF92]).

All these specification methods are based on logic, set theory, algebra, rewriting systems or lambda calculi, all of which have a rich mathematical theory. However, this is also a drawback. To be able to use these formalisms an intensive formal training is required. Another drawback of these formalisms is that, in general, specifications are not executable.

As a reaction to this, in [Gur88, Gur91, Gur95] the logician Yuri Gurevich proposed a new specification method called Evolving Algebras\(^1\). This method is not directly based on the formalism of logic, but on first-order structures (algebras). In fact, this method, like VDM and Z, state oriented. Gurevich claims that “One does not need a Ph.D. in logic to deal with this kind of semantics” [Gur91]. Put differently, it is claimed that this formalism is much easier to use than other methods.

The fact that researchers have succeeded in providing quite a number of case studies using evolving algebras, has given strong evidence to this claim. These studies include, for instance, an algebraic-operational semantics of full PROLOG (see [Bö90a, Bö90b, Bö92, BR94]), and specifications of real-time distributed algorithms (see [GM95]). Also, evolving algebras have been constructed to formalize abstract machine models (see [BR92]), and even to specify computer hardware components (see [BCGR94]). These applications show that the formalism of evolving algebras is not only very simple, but also applicable to a great variety of application areas. Also of importance is the fact that evolving algebras are directly executable, if certain conditions are fulfilled.

The evolving algebra community is growing steadily. Until now, contributions to this research have mostly been on the application side. Less attention has been paid to the theory of evolving algebras. Yuri Gurevich has the conviction\(^2\) that before developing the formal aspects of evolving algebras still more work should be done with respect to the applications. In this spirit the author of this thesis supervised the MSc project of his student Joost Visser during which an evolving algebra interpreter called EVADE was developed and implemented in the typed functional language GOFER (see [Vis96]). This project also resulted in a joint report about the pragmatics of developing evolving algebras (see [TV96]).

As mathematical logic became increasingly important in computer science — for instance in the areas of specification languages, correctness calculi, logic programming, relational databases, and artificial intelligence——, logicians took up again the study of typed logics. This kind of logics has its roots in the first decade of this century, when Bertrand Russell conceived his theory of types to avoid the Russell paradox. In this time, by the way, computers did only exist in the minds of the very imaginative. Moreover, Russell’s type theory was not designed to be used by computer scientists either.

---

\(^1\)Gurevich now prefers to call these Abstract State Machines. In this thesis we will use the old terminology.

\(^2\)Personal communication, 11 September 1995.
A special kind of typed logic is many-sorted logic. The first papers in which the term many-sorted appears, were published in the fifties. Notwithstanding the fact that many-sorted logic is reducible to one-sorted logic, this kind of logic can be made to fit the application at hand more easily. This is not only true for mathematics, but also for computer science. For a good overview of many-sorted logic and its advantages the reader might consult [Man93].

An important application area of many-sorted logic is logic as a tool to reason about programs. Several kinds of logic have been invented to reason about the partial and total correctness of computer programs. Examples include the Hoare-Floyd logics (see e.g. [AO91b]) and dynamic logics which are based on modal logic (see e.g. [Har79]).

In this thesis evolving algebras and many-sorted logic come together. Being aware of the fact that until now almost no attention has been paid to the foundations of evolving algebras, we set out a research program towards a first formal theory about evolving algebras. This theory had to encompass two important aspects.

Firstly, we wanted to formalize the syntax and operational semantics of many-sorted evolving algebras, as opposed to unsorted evolving algebras. Note that, once the syntax and semantics for the many-sorted case have been defined, the syntax and semantics for unsorted evolving algebras directly follow from these as a special case. On the other hand, it is reasonable to formalize these, due to the success of typed programming languages. Moreover, in many papers produced by the evolving algebra community, a many-sorted version of evolving algebras is employed.

There are two ways to formalize these: (i) regard the syntax of many-sorted evolving algebras as a sugared version of unsorted evolving algebras and formalize the latter, which is the current approach taken by Gurevich (see [Gur97]), or (ii) formalize many-sorted evolving algebras directly, which has been our approach. The choice for possibility (ii) implied that certain language constructs could not be reduced to more basic ones. Consequently, these had to be introduced as basic constructs themselves.

Secondly, besides defining their syntax and semantics, we also wanted to develop a Hoare-Floyd-style proof system for evolving algebras. To this purpose we have chosen as meta-logic a many-sorted two-valued logic of partial terms, called E-logic. This logic is extensively studied as well in this thesis. Although the syntax of evolving algebras employed in this thesis is not as rich as the one proposed in [Gur95], our version is rich enough to account for the evolving algebras described, for instance, in [BR94].

This thesis contains five chapters which are about evolving algebras and its meta-logic. The reader is introduced in the realm of evolving algebras in Chapter 2. This chapter contains an overview about the use of evolving algebras and justifies the study carried out in this thesis. Moreover, a tutorial introduction is given to the subject. In the introduction an example is given of a simple evolving algebra that specifies a stack based string reverse algorithm. This
sample evolving algebra will be used throughout the different chapters of the thesis as running example.

In Chapters 3 and 4 the meta-logic is treated. This logic, called $E$-logic, is classically two-valued. It deviates from classical logic in three respects. Firstly, it allows for empty domains; secondly, terms can be non-denoting; and finally, it has a more general notion of substitution which can produce, what we call, extended function symbols and extended terms. It appears that this logic is suitable to reason about evolving algebras.

Our version of $E$-logic is derived from the logic with existence predicate introduced by Dana Scott (see [Sco67, Sco79]). We provided this logic with a Fitch-style deduction system. Moreover, we proved soundness and completeness of this logic in the version with extended function symbols and extended terms. We also defined a number of model theoretic operators, which we have used in Chapter 5 to define the operational semantics of evolving algebras, and we proved some results about them.

Chapter 5 deals with the syntax and semantics of evolving algebras. The syntax is partly based on the syntax of $E$-logic. Most effort of this chapter has been put into the formalization of the semantics of evolving algebras. Before we could do that properly, we had to define the notion of a consistent evolving algebra. The operational semantics has been defined using the ideas of structured operational semantics pioneered by Gordon Plotkin (see [Plo83]).

Evolving algebras employ a kind of fine-grained parallelism, which poses the problem of how to formalize the effect of parallel actions. In the semantics we propose, we solved this problem by reducing the effect of a set of parallel actions to the effects of a corresponding series of actions. Since a semantics is worthless without justification, we analyzed it and proved quite a number of adequacy results about it. These involve renaming bound variables, normal forms, isomorphisms, soundness and completeness. Proving adequacy results for a semantics is useful, since sometimes one is forced to redefine the semantics for reasons of unexpected failures.

Chapter 6 brings together logic and computation. In this chapter we introduce a Hoare calculus, also called axiomatic semantics, for evolving algebras. Some of the rules are straightforward adaptations from those for programs having the WHILE-construct. The other rules, however, are complex. This stems from the fact that evolving algebras accommodate dynamic data structures and operations on these. These can be compared to pointer structures and the operations ALLOCATE and DEALLOCATE, as for example in MODULA 2. Although these kind of structures and operations can be considered as having a low level of abstraction (almost machine level), in the case of evolving algebras this is certainly not true. Having an abstract, yet simple mechanism of dynamic data and operations on them, is one of the appealing features of evolving algebras. At the end of the chapter we prove the correctness of the string reverse algebra of Chapter 2.
The research program set out at the beginning of this study—the formalization of the syntax, operational semantics, and proof theory of many-sorted evolving algebras—can be considered completed. This is not to say that this study is complete. There remain many interesting questions. For example, how to formalize the operational and axiomatic semantics of evolving algebras with external functions and declared variables. Another interesting extension involves multi-agent or distributed evolving algebras (see [Gur95] for definitions of these extensions). Furthermore, the semantics and pragmatics of modularized evolving algebras could be studied. Some proposals of this notion have been made in [Vis96].

The complexity of the correctness proof at the end of Chapter 6 might shock the reader. In that case, we would like to repeat our main goal, which was to provide evolving algebras with a firm foundation. We have shown that this is possible. For a proof system, like the one we have developed, to become useful in practice, a lot of work still has to be done. Without the help of a proof assistant, i.e. a computer program which assists the user in finding proofs, constructing correctness proofs remains a complicated task.
Chapter 1. Introduction and summary
Chapter 2

Introduction to Evolving Algebras

The purpose of this chapter is to introduce the reader into the realm of Evolving Algebras. This is done in two ways. In Section 2.1 we give a sketch of the research in the field, and in Section 2.2 we informally describe what an Evolving Algebra is and how it operates.

2.1 Why Evolving Algebras?

In his paper [Gur88], and later more elaborated in [Gur91] and [Gur95], the logician Yuri Gurevich proposed a new model of computation: Evolving Algebras. These algebras can be considered as highly abstract machines which can simulate algorithms step by step. The underlying principle which makes this possible is the fact that the ‘data types’ and ‘machine instructions’ of an evolving algebra can be tailored to the abstraction level one wishes to have. This clearly is an advantage to the machine models proposed in the past, the instruction sets and/or data types of which are fixed and of a low abstraction level. This does not only hold for Turing Machines, but also for the Schönhalge Storage Modification Machines (see [Sch80]) and the Kolmogorov–Uspensky Machines (see [KU58]), which do resemble evolving algebras.

In the almost ten years that evolving algebras have been studied, more than ten application areas have been found. The web site on evolving algebras [Hug97] lists the following application areas: architectures, benchmark examples, compiler correctness, databases, distributed systems, interpreters, logic, natural languages, programming languages, real-time systems, and WAM/logic programming. In the following two paragraphs we will list some examples.

With respect to architectures, evolving algebra specifications have been developed for parallel architectures like the APE100 (see for example [BCGR94]). An interesting benchmark example is about a specification and verification of a railway crossing system (see [GH96]), which is modeled by a distributed evolving algebra. Compiler correctness has been demonstrated, for instance, for the

\footnote{Nowadays Evolving Algebras are also known as Abstract State Machines.}
Chapter 2. Introduction to Evolving Algebras

Warren Abstract Machine in [BR94]. A nice example of a real-time distributed evolving algebra for the Group Membership Protocol can be found in [GM95].

In [PH94] a method is described to derive partial correctness logics from evolving algebras modeling the semantics of sequential, deterministic programming languages. An evolving algebra abstract machine is defined in [CDG96] in terms of an universal evolving algebra. In Chapter 1 we already mentioned the impressive work done by Börger and Rosenzweig regarding the formalization of full PROLOG (see [Bör90a, Bör90b, Bör92, BR94]). Other programming languages the operational semantics of which has been formalized by evolving algebras are, for example, MODULA 2, OCCAM, and PARLOG (see [GM88, Mor88, GM90, BR93]).

In fact, the mentioned web site [Hug97] contains more than 60 papers on evolving algebras and applications of them. So, the question arises "Why are evolving algebras so interesting?" According to [Hug97], evolving algebras have the following nice features:

**Precision** Evolving algebras offer a simple syntax coupled with a rigorous, well-understood mathematical semantics based on first-order structures.

**Faithfulness** It is possible to adjust the abstraction level of evolving algebras to the application. In this way difficult notations are avoided.

**Understandability** Since the syntax and semantics of evolving algebras are simple, and since the abstraction level of the evolving algebra is tuned to the application, specifications are easy to understand.

**Executability** Evolving algebras are executable if the built-in operations are computable. This makes the evolving algebra methodology usable for checking specifications and prototyping.

**Scalability** The possibility to play with the abstraction level of evolving algebras makes it possible to specify on different levels of abstraction. This enhances top-down design.

**Generality** The evolving algebra methodology has proven to be applicable to many areas, ranging from architectures to natural language. Other methodologies, like various temporal logics, do not possess this feature.

The numerous papers on evolving algebras show, to a certain extent we believe, that Gurevich's claim that "One does not need a Ph.D. in logic to deal with this kind of semantics" can be justified. This is not say that specifications are easy to read. However, we don't believe that the evolving algebra formalism is causing this. Instead, we think that evolving algebras show the inherent complexity of a problem domain. This means that an evolving algebra is difficult to the extent that its application domain is difficult.

There are other algebraic specification languages which can be used to describe the semantics of various objects. Most of these languages are purely equational.
2.1. Why Evolving Algebras?

One of the exceptions to this is the language COLD-K, which is also state-based, the states being algebras (see [FJ92]). The logic employed in COLD-K is, like the one we propose for evolving algebras, a two-valued many-sorted logic of partial functions. Its proof system is based on dynamic logic, whereas we use a Hoare-style calculus for evolving algebras. Since this specification language combines algebraic and state-based specification styles, its syntax is fairly complicated.

Another approach which resembles evolving algebras is the language UNITY described in [CM88]. A program in UNITY is a set of multiple, possibly conditional, assignments from which one is nondeterministically chosen to be applied during each stage in a run. The nondeterministic selection of assignments is constrained by a fairness principle. This language was designed as a foundation for parallelism. Evolving algebras have, as we have seen, a much broader scope.

The motivation for this study was twofold. Firstly, we wanted to rigorously formalize the notion of a many-sorted evolving algebra. The papers on evolving algebras we have seen so far, did not carry out such a formalization. A possible exception is [GR93], but this paper is about one-sorted evolving algebras. Secondly, in [BR92] we found the following remark on page 3: "[...] useful deductive systems for evolving algebras have yet to be developed." In the remainder of this section we will describe the relation of our work to work done by Gurevich and others.

The original presentation of evolving algebras by Gurevich makes use of one-sorted static algebras, in which all used data types are subsets of the so-called superuniverse of the algebra. As a consequence, data types are identified by their characteristic functions.

The presentation in this study diverges from this way of handling data types. We use many-sorted partial algebras. This situation can be compared to the distinction between one-sorted logic and many-sorted logic: here many-sorted logic can be translated to one-sorted logic (see [Man93]). Although this is possible, in practice it is often more convenient to have the different data types beforehand, as this appeals more to the intuition of the user.

So, the syntax we are proposing for evolving algebras is many-sorted in nature. As a consequence, data types cannot be described anymore by their characteristic functions. In order to describe so-called universe extensions and universe contractions, we therefore need basic language constructs. In [Gur95] there is no need for a separate language construct for universe contractions: an element can be removed by simply setting its characteristic function to false.

With respect to universe extensions, Gurevich uses a special language construct, called import. With the help of this construct and a characteristic function, a universe extension can be expressed by binding the imported variable to a new element and setting the characteristic function for this variable to true. To simplify notation, he then introduces a shorthand, the syntax of which slightly differs from our syntax for universe extensions. However, the
formulation of the semantics is quite different from ours and less transparent, we believe.

Another difference with the syntax proposed in [Gur95] concerns the nesting of rules. Our syntax is much more restricted, but rich enough to model, for example, the evolving algebras in [BR94]. The rules in [Gur95] can be arbitrary nested. Moreover, in that paper also evolving algebras with external functions (which are set by the environment) and declared variables (which enable massive local parallelism) are defined. Further extensions of the formalism involve evolving algebras with nondeterministic choice, and evolving algebras with more than one agent. The latter so-called distributed evolving algebras can be used to model distributed algorithms.

The semantics of our evolving algebras is expressed in terms of partial many-sorted algebras the universes of which may possibly be empty. We distinguish static and dynamic universes, only the latter of which can be changed during run time. The static universes are to be understood as standard sets, like the set of natural numbers. This feature allows to tailor the abstraction level of the evolving algebras to the level needed. Dynamic universes can be used as an algebraic model for dynamic structures which, for instance, are created in runtime by computer programs containing pointer types and memory allocation statements.

In order to express the dynamic behaviour of evolving algebras on static partial many-sorted algebras we have chosen the formalism of Structured Operational Semantics (for an introduction, see e.g. [Hen90, NN92]). This operational semantics reflects the block structure which is introduced by our syntax to express universe extensions. Also, the semantics is such that the effect of parallel updates can be described serially. The informal semantics in [Gur95] is based on the idea of collecting all updates in a rule and applying these simultaneously. This leads, in the case of import constructs, to complicated constructions.

We believe that our approach is also applicable to the other language constructs defined in [Gur95]. But this would be a topic of future research. All syntactic and semantic notions are defined in Chapter 5. In that chapter we will also prove that the proposed operational semantics is adequate.

The work done in this study is also related to [GR93]. However, the starting point in that paper is the one-sorted approach and the emphasis is on concurrency. The formalism used is highly inspired by dynamic logic (see [Har79, Har84]). Our focus was on developing a proof calculus for evolving algebras. We believe that a Hoare-style calculus is more apt for correctness arguments than dynamic logic. This claim can be substantiated by referring to the fact that many course books on program correctness take the Hoare-like approach (see e.g. [AO91a, AO91b, Dah92, Fra92]).

Chapter 6 deals with a Hoare-style calculus to reason about evolving algebras. How this calculus can be used to prove the (partial and total) correctness of an evolving algebra, will be demonstrated for the sample evolving algebra
2.2. What is an Evolving Algebra?

we will present in Example 2.2.2. We also will prove the correctness of this calculus. The calculus will turn out to be provably incomplete, however.

We will not study the expressive power of evolving algebras nor their time complexity. One thing is clear, however. All computable functions can be computed by an evolving algebra. Also, universal evolving algebras exist which can simulate evolving algebras (see [BG94]).

2.2 What is an Evolving Algebra?

The idea of an evolving algebra is as follows. An evolving algebra is a finite set of rules which induces a computation on a given initial state. The idea of a state is not new, of course. A state can be seen as a snapshot of a computer memory. It associates identifiers with values. In our case, a state also associates functions with their graphs. More abstractly, a state can be seen as an algebra. Since an algebra is a static mathematical object, Gurevich calls it a static algebra as opposed to an evolving algebra.

A computation is a sequence of static algebras as induced by the rules of the evolving algebra. Put in other words, a computation consists of modifying static algebras in a number of computation steps. This explains the name evolving algebra: during a computation the initial algebra evolves towards a final state or algebra, in which no rule is longer applicable. Of course, a final state does not always exist.

A rule of an evolving algebra consists of a finite number of instructions or updates. These updates actually modify an algebra. Not only can it change the association of identifiers with values, it also can change the association of functions with their graphs.

If one chooses to work with many-sorted static algebras, it is possible to decide that the universes or universes of the algebra may also change in time, i.e. during a computation. This motivates the idea of distinguishing between static and dynamic objects of an algebra.

The first class of objects are those universes, constants and functions which are not allowed to change in time, and the second class consists of those which may. Static objects can be seen as the 'built-ins' of an evolving algebra. Put in another way, the static objects represent the standard operations which an evolving algebra is to provide to the user. Examples of static objects are the set of natural numbers, together with the constant 0, the successor function $S$ and the addition $+$. It would be pointless to change the interpretation of these static objects during a computation.

The second class of objects are used to store values which are used in the computation, just like in the case of imperative programs. However, dynamic functions can be used to mathematically model the notion of a pointer. From an abstract point of view, a pointer is nothing else than the value of a function: $f(a) = b$ expresses that $f$ is a pointer from object $a$ to object $b$. 

In order to describe a static algebra, one needs the notion of a *signature*. A signature specifies the logical form of an algebra, i.e. which universes, constants and functions it consists of. Moreover, for each of this kind of objects, it specifies whether the object is static or dynamic.

### 2.2.1 Example

An example of a signature would be:

- **Static universes**: \( \text{Char}, \text{Char}^*; \)
- **Dynamic universe**: \( \text{Stack}; \)
- **Static constant**: \( \text{nil}; \)
- **Dynamic constants**: \( \text{bottom, top}; \)
- **Static functions**: \( \text{head, tail, append}; \)
- **Dynamic functions**: \( \text{input, output, next, content}; \)

where the static and dynamic constants and functions have the following types:

- \( \text{input, output} : \text{Char}^*; \)
- \( \text{head} : \text{Char}^* \rightarrow \text{Char}; \)
- \( \text{tail} : \text{Char}^* \rightarrow \text{Char}^*; \)
- \( \text{nil} : \text{Char}^*; \)
- \( \text{append} : \text{Char}^* \times \text{Char} \rightarrow \text{Char}^*; \)
- \( \text{top, bottom} : \text{Stack}; \)
- \( \text{next} : \text{Stack} \rightarrow \text{Stack}; \)
- \( \text{content} : \text{Stack} \rightarrow \text{Char}. \)

The intended semantics of the elements of the signature is that \( \text{Char} \) refers to some fixed set of characters, \( \text{Char}^* \) to the set of all finite strings over \( \text{Char} \), and that \( \text{Stack} \) is representing a stack of characters. The identifier \text{input} stores the current value of the input, which initially is the string to be read. Likewise \text{output} refers to the current output string. The names \text{bottom}, \text{top} and \text{next} provide the stack structure for the set denoted by \( \text{Stack} \) in the following way: \( \text{bottom} \) denotes the bottom element of the stack, \( \text{top} \) the top element, and \( \text{next}(a) = b \) if the stack element \( a \) is on top of the stack element \( b \). Finally, the function \text{content} maps stack elements to their contents, which is, in this case, a character.

The other constants and functions are held constant during each computation. They refer to the standard string operations, as for example known in functional programming. In this example, the choice for the signature and these static objects is motivated by the string reverse algorithm below (see Example 2.2.2).

With the help of a signature, the rules of an evolving algebra can be described. As is explained above, the rules constitute the mechanism which changes static algebras in time. These rules consist of a guard and a set of updates which
have to be carried out on the current state if the guard is evaluated true in that current state. Nondeterminism enters here in those cases where the guard of more than one rule becomes true. In such cases one of the applicable rules is selected nondeterministically.

There are three kinds of updates. The most basic of these is the function update. If, for a given signature, \( c \) is a variable or a dynamic constant, and \( t \) is a term, then:

\[
c := t
\]

is called a local (function) update. Executing it on the current state has as result that the value of \( c \) in the next state will be the value of \( t \) in the current state. More abstractly, the update \( c := t \) transforms the current state into a new state, in which the denotation of \( c \) has been changed into the current denotation of \( t \).

Likewise, if \( f \) is a dynamic function, and \( s_1, \ldots, s_n, t \) are terms, then:

\[
f(s_1, \ldots, s_n) := t
\]

is also a function update. The effect of this update is that the denotation of \( f \) in the next state is equal to its denotation in the old state, except that the function value on the current values of \( s_1, \ldots, s_n \) is changed into the current value of \( t \). Note that this update also transforms the current state into a new state.

Updates of this kind are only defined if the terms \( s_1, \ldots, s_n \) do have a value in the current static algebra. However, the value of \( t \) need not be defined. This means that the static algebras we consider are partial.

The second kind of update is the contraction update:

\[
\text{rem } t : u,
\]

which amounts to deleting the element denoted by \( t \) in the current state from the universe denoted by \( u \). In this case the value of \( t \) has to be defined before the execution of the contraction. So, in the new state the universe denoted by \( u \) does not contain the element associated with \( t \) in the old state.

The last kind of update is the extension update. This kind of update introduces a kind of ‘block’ structure. It is of the following form:

\[
\text{new } x : u \text{ with } \mathcal{U},
\]

where \( x \) is a variable of type \( u \) and \( \mathcal{U} \) is a finite set of updates. Its effect is that the universe denoted by \( u \) is extended by a new element which is temporarily bounded to \( x \) and for which the updates in \( \mathcal{U} \) are executed. This means that all variables \( x \) occurring free in \( \mathcal{U} \) are bound. Note that \( \mathcal{U} \) may contain extension updates. Here the mentioned block structure enters the picture. After completion of this update, \( x \) is reset to its value in the old state, if it still exists, and otherwise it will be undefined.
Now we can introduce the notion of a rule. A rule is a construct:

\[
\text{if } \varphi \text{ then } \mathcal{U},
\]

where \( \varphi \) is a quantifier free formula, called its guard, and \( \mathcal{U} \) is a finite set of updates. An evolving algebra is a non-empty, finite set of rules. The effect of an evolving algebra on an initial algebra \( \mathcal{A}_0 \) is a sequence

\[
\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m, \mathcal{A}_{m+1}, \ldots, \mathcal{A}_n, \ldots
\]

of algebras, also called a run, such that every \( \mathcal{A}_{m+1} \) is the result of applying one of the rules of the evolving algebra to \( \mathcal{A}_m \). A rule can only be applied to \( \mathcal{A}_m \) if its guard is true in \( \mathcal{A}_m \). The application of a rule consists of a parallel execution of the updates occurring in it. If the rules contain free variables, then a run is a sequence of interpretations, i.e. pairs of partial algebras and partial variable assignments.

If more than one rule is applicable to \( \mathcal{A}_m \), i.e. if more than one guard of the rules of the evolving algebra evaluate to true in \( \mathcal{A}_m \), then one of them is nondeterministically chosen. The computation terminates with result \( \mathcal{A}_n \) if no guard of some rule evaluates true with respect to \( \mathcal{A}_n \). Of course, there exist evolving algebras which induce non-terminating runs.

A proviso must be made as to the application of a rule. Therefore, we introduce the notion of being a consistent rule with respect to a given algebra. Informally, a rule is consistent with respect to an algebra if it is not the case that some updates in that rule try to set the value of a function to different values for the same arguments, or try to delete an element of a universe while at the same time setting the value of a function to this element or changing the value of a function having this element amongst its arguments. The proviso now is, that a rule can only be applied to an algebra if the guard of that rule is true and if the rule is consistent with respect to that algebra.

2.2.2 Example The following evolving algebra is a formalization of a string reverse algorithm. It is based on the signature introduced in Example 2.2.1. When trying to understand the way this evolving algebra works, the reader should recall that all updates of a rule are executed in parallel.

\[
\text{if } \text{input } \neq \text{nil then}
\]

\[
\text{new } x : \text{Stack with}
\]

\[
\begin{align*}
\text{top} & := x \\
\text{next}(x) & := \text{top} \\
\text{content}(x) & := \text{head(input)} \\
\text{input} & := \text{tail(input)}
\end{align*}
\]

\[
\text{if } \text{input } = \text{nil } \land \text{top } \neq \text{bottom then}
\]

\[
\begin{align*}
\text{output} & := \text{append(output, content(top))} \\
\text{top} & := \text{next(top)} \\
\text{rem} \text{top} & : \text{Stack}
\end{align*}
\]
2.2. What is an Evolving Algebra?

Clearly, this evolving algebra consists of two rules the guards of which cannot both be true in any algebra for the signature. It is also intuitively clear that both rules are consistent with respect to any algebra which occurs in a run of the evolving algebra.

The operation of the evolving algebra is as follows. The dynamic constant input is assumed to have some initial value in the initial algebra from which the computation starts. Moreover, it is required that the Stack is empty in the initial algebra. This is realized by letting top and bottom denote the same element of the Stack.

Then, until the value of input is the empty string nil, the first rule is applied several times, where each time the head character of input is being read and being pushed on the Stack. Pushing is implemented by creating a new element of type Stack, letting this element be the new top, letting the content of this new top be the read character, and making a 'pointer' from the new top to the old one by setting the function next.

When the first rule is no longer applicable, the second one comes into action. It is executing as long as the Stack is not empty. Any application of the rule consists of popping an element from the Stack and appending it to the dynamic constant output. Popping consists of taking the current top element, taking its content, removing the current top, and setting the new top to the next element in the Stack. If the Stack has been read out, the constant output contains the reverse of the original input string. In that case none of the two rules is applicable any more and the computation stops.

Note that the contraction update rem top : Stack in the second rule does not contribute to the correct operation of the algorithm. It is only there for 'garbage collection'.

In Chapter 6 we will be able to prove that this evolving algebra is partially and totally correct with respect to some reasonable precondition and postcondition. This exercise will not turn out to be simple, however.
Chapter 2. Introduction to Evolving Algebras
Chapter 3

The syntax of $E$-logic

In order to reason about partially defined functions—which we will do all the time when using Evolving Algebras—we need a logic of partial functions. There are several formalisms suggested in the literature. For an overview see e.g. [Che86]. One of the first formalizations is by Scott. The logic he proposed in [Sco67] is a one sorted classical logic extended with the $E$ predicate, where $E(t)$ means that the term $t$ exists, i.e. is denoting. In [Sco79] the formalization is carried over to intuitionistic logic. The name $E$-logic is used for both versions of the formalism, see e.g. [RdL84, TD88, Kup94].

In this thesis we will use a many-sorted classical version of $E$-logic with additional strictness conditions. Moreover, our logic deals with, what we call, extended function symbols and terms. The idea behind extended function symbols is that they model locally updated functions (cf. array-assignments in programming languages). Extended terms differ from ordinary ones in that they may contain extended function symbols.

For this logic we will provide a Fitch-style natural deduction system (see [Fit52, WST93]) and prove a completeness result by means of the well-known Lindenbaum-Henkin construction.

This chapter is organized as follows. In Section 3.1 all basic syntactical notions will be introduced. Then, in Section 3.2, the topic of substitution will be covered. Finally, in Section 3.3 the Fitch-style deduction system will be described.

3.1 Signatures, terms, and formulae

We start this section by giving the obvious definitions. The reason to give them anyway, is that we like to give a complete account of the $E$-logic we will be using as our assertion language, i.e. the language in which we will formulate and prove properties of evolving algebras.

3.1.1 Convention If $V$ is a set, then $V^*$ denotes the set of all strings over $V$. Likewise, $V^+$ denotes the set of all non-empty strings over $V$. Elements of $V^*$ or $V^+$ are written as $v_0 \ldots v_n$ where $v_i \in V$ ($0 \leq i \leq n$). The empty string is represented by $\lambda$. (In the literature $*$ is known as the Kleene star.)
In the next definition the notion of a *signature* for E-logic will be introduced. A signature fixes what is also called the ‘non-logical alphabet’ of a language. In our case a signature is a quadruple consisting of a special symbol $\Omega$, a set of *sort symbols*, a set of *non-logical symbols*, and a function which assigns arities to the symbols with respect to the sort symbols.

The symbol $\Omega$ will be used to denote the ‘sort’ of predicates. Although functions may be nullary — these kind of functions being normally called *(individual) constants* —, predicates are required to have arguments. This will be evident from the definition of the arity function below.

### 3.1.2 Definition

*A signature $\Sigma$ for a language of E-logic is a quadruple*

$$\Sigma = \langle \Omega, \text{SORT}, \text{SYM}, \sigma \rangle,$$

*such that:*

1. **SORT** and **SYM** are finite sets of symbols, where $\Omega \notin \text{SORT}$,
2. $\sigma$ is a function

$$\sigma : \text{SYM} \rightarrow (\text{SORT}^* \times \text{SORT}) \cup (\text{SORT}^+ \times \{\Omega\}),$$

*called the **arity function**.*

### 3.1.3 Convention

i. In a context where different signatures are being used we will index the names of symbol sets by the name of the appropriate signature, e.g. $\text{DSORT}(\Sigma)$. The same applies to the function $\sigma$ and the various sets to be defined below.

ii. Variables ranging over **SORT** are denoted by: $u, u_0, u_1, \ldots, v, \ldots$

iii. Variables ranging over **SORT**$^*$ are denoted by: $w, w_0, w_1, \ldots$

iv. Elements $c$ of **SYM** for which $\sigma(c) = (\lambda, u)$ for some $u \in \text{SORT}$ are called **individual constants**. The set of individual constants with arity $(\lambda, u)$ is denoted by $\text{CON}_u$.

v. Elements $f$ of **SYM** for which $\sigma(f) = (w, u)$ for some $w \in \text{SORT}^+$ and $u \in \text{SORT}$ are called **function symbols**. The set of function symbols with arity $(w, u)$ is denoted by $\text{FUN}_{w,u}$.

vi. Elements $P$ of **SYM** for which $\sigma(P) = (w, \Omega)$ for some $w \in \text{SORT}^+$ are called **predicate symbols**. The set of predicate symbols with arity $(w, \Omega)$ is denoted by $\text{PRED}_w$.

vii. By abuse of language elements of **SORT**, $\text{CON}_u$, $\text{FUN}_{w,u}$, and $\text{PRED}_w$ will often be called sorts, constants, functions and predicates.

viii. Variables ranging over $\text{CON}_u$ are denoted by $a, b, \ldots$
3.1. Signatures, terms, and formulae

ix. Variables ranging over $\text{FUN}_{w,u}$ are denoted by $f, g, \ldots$.

x. Variables ranging over $\text{PRED}_{w}$ are denoted by $P, Q, \ldots$.

To any $\Sigma$ a first-order many-sorted language $\mathcal{L}_{\Sigma}$ can be associated. How this can be done is subject of the following definitions.

3.1.4 Definition Associated with a signature $\Sigma$ with sorts $\text{SORT}$, a family of mutually disjoint, countably infinite sets $\text{VAR}_u$ of variables of sort $u$ for each $u \in \text{SORT}$ is given. The sets $\text{VAR}_u$ are disjoint from the set $\text{SYM}$.

3.1.5 Convention Elements of $\text{VAR}_u$ are denoted by: $x^u, x_0^u, x_1^u, \ldots, y^u, \ldots$.

It would have been possible to include the variables in the signature. However, we feel that variables do not have real meaning: they are binders, sometimes called 'place holders'. In this vision, variables are of a logical nature and do not belong to the non-logical alphabet of a language.

In the next definitions we will define the sets $\text{TERM}_u$ of terms of sort $u$, and the set $\text{FORM}$ of formulae with respect to a signature $\Sigma$. Every set $\text{TERM}_u$ will contain a special element $\perp_u$ which will figure as the 'undefined value of sort $u$'. We will mostly suppress the subscript $u$ in $\perp_u$.

We will also define the sets $\text{XFUN}_{w,u}$ and $\text{XTERM}_u$ of extended function symbols with arity $(w,u)$ and extended terms of sort $u$, respectively. Because in evolving algebras functions can be subject to updates, we want to be able to reflect this in the syntax of the assertion language by means of ordinary substitutions and array substitutions (see for example [AO91a]). Extended function symbols will then come into existence as the result of substitutions.

The notation we will adopt, is as follows. If $f$ is a function symbol, then the expression $f[r \mapsto s]$ is an extended function symbol. The meaning of this extended function symbol is that $f[r \mapsto s](x)$ evaluates to the value of $s$ if $r$ evaluates to the value of $x$, and the value of $f(x)$ otherwise.

This idea of 'updated' function symbols is generalized as follows. The extended function symbol $f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]$ denotes the function that has the value denoted by $s_i$ if its argument is equal to the value denoted by $r_i$ ($1 \leq i \leq m$), otherwise it behaves like the function denoted by $f$. There is one constraint, however, in this case: if for some $i$ and $j$ such that $1 \leq i < j \leq m$ it holds that $r_i$ and $r_j$ evaluate to the same value, but $s_i$ and $s_j$ do not, then the extended function symbol denotes the function which is undefined at the value of $r_i$.

Another form that extended function symbols can have is $f[\ldots] \ldots [\ldots]$. Also, the terms occurring in extended function symbols may contain extended function symbols. For this reason the set of extended terms, that is the set of terms which contain extended function symbols, and the set of extended function symbols are defined mutually recursively.

The notation $f[r \mapsto s]$ was inspired by the notation for map override as used in the specification language VDM (see for example [Jon90]). We use $f[r \mapsto s](t)$
as an alternative to the widely used notation if $t = r$ then $s$ else $f(t)$. In fact our notation is more 'abstract' in the following sense. Using $\lambda$-abstraction, $f[r \mapsto s]$ can be defined as follows:

$$f[r \mapsto s] = \lambda x [if \ x = r \ then \ s \ else \ f(x)].$$

3.1.6 Remark It is in principle possible to have extended constants, like $c[\mapsto s]$, $c[\mapsto s_1, \ldots, \mapsto s_m]$, $c[\ldots][\ldots]$. For the moment we choose not to do so for the following reasons. Extended constants like $c[\mapsto s]$ can immediately be reduced to $s$. Those like $c[\mapsto s_1, \ldots, \mapsto s_m]$ are non-denoting if the $s_i$ denote different values. But substitutions of the form $[c:=s_1, \ldots, c:=s_m]$ can be forbidden on syntactic grounds. The ban on this kind of substitutions then belongs to the static semantics of evolving algebras as will become apparent later in this thesis.

The same convention will apply to substitutions like $[x:=s_1, \ldots, x:=s_m]$. Finally, there seems to be no reason why someone would like to have rules containing two updates of the same constant.

3.1.7 Convention i. Let $w \in \text{SORT}^+$ and $w \equiv u_1 \ldots u_n$, then the name $\text{TERM}^w$ will denote the set $\text{TERM}^w_{u_1} \times \ldots \times \text{TERM}^w_{u_n}$. The same convention will be used for extended terms and variables.

ii. We will use bold math letters to denote vectors of terms, e.g. if $t_i \in \text{TERM}_{u_i}$, ($1 \leq i \leq n$) and $w \equiv u_1 \ldots u_n$ for some positive $n \in \mathbb{N}$, then $t$ denotes the vector $(t_1, \ldots, t_n) \in \text{TERM}^w$. Again, the same convention will be used for vectors of extended terms or variables.

3.1.8 Definition Let a signature $\Sigma$ be given. The sets $\text{TERM}_u$ of terms of sort $u$ are mutually recursively defined to be the smallest sets such that:

1. $\bot_u \in \text{TERM}_u$,

2. $\text{VAR}_u \cup \text{CON}_u \subseteq \text{TERM}_u$,

3. $f \in \text{FUN}_{w,u}$ & $t \in \text{TERM}^w \Rightarrow ft \in \text{TERM}_u$.

The symbol $\bot_u$ is called a logical constant and is chosen to be different from the elements of $\text{SYM}$ and $\text{VAR}_u$ ($u \in \text{SORT}$).

3.1.9 Definition Let a signature $\Sigma$ be given. The sets $\text{XFUN}_{w,u}$ and $\text{XTERM}_u$ of extended function symbols of arity $(w, u)$ and extended terms of sort $u$ are mutually recursively defined to be the smallest sets such that:

1. $\bot_u \in \text{XTERM}_u$,

2. $\text{VAR}_u \cup \text{CON}_u \subseteq \text{XTERM}_u$,

3. $\text{FUN}_{w,u} \subseteq \text{XFUN}_{w,u}$,
3.1. Signatures, terms, and formulae

4. \( f \in \text{XFUN}_{w,u} \quad \& \quad m > 0 \)
   \& \( r_1, \ldots, r_m \in \text{XTERM}^w \quad \& \quad s_1, \ldots, s_m \in \text{XTERM}_u \)
   \( \Rightarrow f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \in \text{XFUN}_{w,u} \),

5. \( f \in \text{XFUN}_{w,u} \quad \& \quad t \in \text{XTERM}^w \quad \Rightarrow \quad ft \in \text{XTERM}_u \).

In the following definition the set \( \text{FORM} \) of formulae is defined. This is done in the usual way. Note that we introduce the propositional constants \( \text{tt} \) and \( \text{ff} \) to denote truth and falsehood, respectively.

Moreover, we have the unary predicate symbol \( \downarrow \) which denotes definedness: for a term \( t \) the expression \( \downarrow t \) will be true in some interpretation if \( t \) is a denoting term, else false. Our notation differs from the one adopted in e.g. [Sco79], where the symbol \( E \) is used in stead of \( \downarrow \). This explains, by the way, the name of our logic: \( E \)-logic.

The symbol \( = \) will be interpreted as existential equality: \( (s = t) \) will be true in some interpretation if both \( s \) and \( t \) are defined and equal when evaluated. In all other cases \( (s = t) \) will be false. In fact this means that \( = \) is a strict binary predicate symbol.

Regarding strictness, if for some term one of its subterms is non-denoting in some interpretation, then the term itself will be non-denoting as well. Furthermore, if for some predicate symbol one of its subterms is non-denoting in some interpretation, then the atomic formula consisting of this predicate symbol and its arguments, will be false in that interpretation. As a consequence of the preceding remarks our logic is two-valued.

Of course it is not necessary to impose strictness conditions of this kind. For example in \( LPF \), the logic being employed for the specification language \( VDM \) (see [Jon90, JM93]), there are no strictness conditions on functions and predicates. For the identity, however, a strictness condition does hold: if one of the arguments of \( = \) is undefined, then the equality itself is undefined. This means that the logic is partial, in this case having three truth values.

3.1.10 Definition Let a signature \( \Sigma \) be given. The set \( \text{FORM} \) of formulae is defined by:

1. \( \text{tt, ff} \in \text{FORM} \),

2. \( t \in \text{XTERM}_u \quad \Rightarrow \quad \downarrow t \in \text{FORM} \),

3. \( s, t \in \text{XTERM}_u \quad \Rightarrow \quad (s = t) \in \text{FORM} \),

4. \( P \in \text{PRED}_w \quad \& \quad t \in \text{XTERM}^w \quad \Rightarrow \quad Pt \in \text{FORM} \),

5. \( \varphi, \psi \in \text{FORM} \quad \Rightarrow \quad \neg \varphi, (\varphi \land \psi), (\varphi \lor \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi) \in \text{FORM} \),

6. \( \varphi \in \text{FORM} \quad \& \quad x^u \in \text{VAR}_u \quad \Rightarrow \quad (\forall x^u \varphi), (\exists x^u \varphi) \in \text{FORM} \).

The symbols \( \text{tt, ff, } \downarrow, =, \neg, \land, \lor, \rightarrow, \leftrightarrow, \forall \) and \( \exists \) are called logical constants and are chosen to be all different from each other and from the elements of \( \text{SYM} \) and \( \text{VAR}_u \) \( (u \in \text{SORT}) \).
We did not explicitly demand the extra-logical symbols (, ), [ , ] and $\mapsto$ to be different from all other symbols, but we will tacitly assume this always to be the case.

3.1.11 Convention i. Extended function symbols are, like non-extended function symbols, denoted by: $f$, $g$, . . . .

ii. Terms are denoted by: $s$, $t$, . . . .

iii. The formula $\neg (s = t)$ will be often abbreviated by $s \neq t$.

iv. Formulae are denoted by Greek letters: $\varphi$, $\psi$, . . . .

v. Sets of formulae are denoted by: $\Gamma$, $\Delta$, . . . .

vi. Superfluous parentheses are suppressed according to standard practice.

vii. Instead of $\forall x^u \varphi$ and $\exists x^u \varphi$, we will also write $\forall x : u \cdot \varphi$ and $\exists x : u \cdot \varphi$, respectively.

In the following definition the sets of free variables in extended function symbols and extended terms are defined in the usual way.

3.1.12 Definition If $f \in \text{XFUN}_{w,u}$ and $t \in \text{XTERM}_u$ for some $w \in \text{SORT}^+$ and $u \in \text{SORT}$, then the sets $\text{FV}(f)$ and $\text{FV}(t)$ of free variables in $f$ and $t$, respectively, are mutually recursively defined by the following equations (where the corresponding cases of Definition 3.1.9 are considered in the same order):

$\text{FV}(\bot) = \emptyset$,

$\text{FV}(x^w) = \{x^w\}$,

$\text{FV}(c) = \emptyset$,

$\text{FV}(g) = \emptyset$,

$\text{FV}(g|r_1 \mapsto s_1, \ldots , r_m \mapsto s_m|) = \text{FV}(g) \cup \bigcup_{i=1}^{m}(\text{FV}(r_i) \cup \text{FV}(s_i))$,

$\text{FV}(gt) = \text{FV}(g) \cup \text{FV}(t)$.

In this equations, $\text{FV}(t)$ for a vector $t = (t_1 , \ldots , t_n) \in \text{XTERM}_w^u$, is defined by:

$\text{FV}(t) = \bigcup_{i=1}^{n}\text{FV}(t_i)$.

Next, we will extend the definition of the function Fv to formulae and sets of formulae. The definition is straightforward.

3.1.13 Convention We will use $*$ and $Q$ as metavariables ranging over the sets $\{\land , \lor , \rightarrow , \leftrightarrow\}$ and $\{\forall, \exists\}$, respectively.
3.2. Substitution

3.1.14 Definition If $\varphi \in \text{FORM}$ then the set $\text{Fv}(\varphi)$ of free variables in $\varphi$ is recursively defined by:

\[
\begin{align*}
\text{Fv}(\mathbf{tt}) &= \emptyset, \\
\text{Fv}(\mathbf{ff}) &= \emptyset, \\
\text{Fv}(\mathbf{tt}) &= \text{Fv}(t), \\
\text{Fv}((s = t)) &= \text{Fv}(s) \cup \text{Fv}(t), \\
\text{Fv}(\mathbf{Pt}) &= \text{Fv}(\mathbf{t}), \\
\text{Fv}(\neg \psi) &= \text{Fv}(\psi), \\
\text{Fv}((\psi \ast \chi)) &= \text{Fv}(\psi) \cup \text{Fv}(\chi), \\
\text{Fv}(\forall x^{u} \psi) &= \text{Fv}(\psi) \setminus \{x^{u}\}.
\end{align*}
\]

If $\Gamma \subseteq \text{FORM}$, then the set $\text{Fv}(\Gamma)$ of free variables in $\Gamma$ is defined by:

\[
\text{Fv}(\Gamma) = \bigcup_{\psi \in \Gamma} \text{Fv}(\psi).
\]

3.1.15 Definition Let a signature $\Sigma$ be given. The set Sent of sentences is defined by:

\[
\text{Sent} = \{ \varphi \in \text{FORM} \mid \text{Fv}(\varphi) = \emptyset \}.
\]

3.2 Substitution

In the next definition we will define general substitution. This kind of substitution not only makes it possible to substitute terms for variables, but also terms for terms. This latter feature changes the interpretation of function symbols. For example, the substitution $[f(a, b) := c]$ changes the interpretation of the function symbol $f$: its value for the arguments determined by $a$ and $b$ is set to the actual value of $c$, with respect to some interpretation. Such an update is called a function update.

The version of general substitution we will describe, is parallel. So our substitutions will have the general outline $[s_{1} := t_{1}, \ldots, s_{m} := t_{m}]$ for some positive $m \in \mathbb{N}$.

3.2.1 Convention The symbol $\equiv$ will be used to express syntactic equality between expressions.

3.2.2 Definition Let $s, t \in \text{TERM}_{u}$, and let $(s_{1}, \ldots, s_{m}) \in \text{TERM}^{w}$ and $(t_{1}, \ldots, t_{m}) \in \text{TERM}^{w}$ be such that $s_{i} \neq s_{j}$ whenever $i \neq j$. Then $[s := t]$ is called a simple substitution. Furthermore, the set of all simple substitutions $[s_{i} := t_{i}]$ (where $1 \leq i \leq m$), notation $[s_{1} := t_{1}, \ldots, s_{m} := t_{m}]$, is called a substitution.
3.2.3 REMARK The order of the simple substitutions \([s_i := t_i]\) in the substitution \([s_1 := t_1, \ldots, s_n := t_n]\) is immaterial as substitutions are defined to be sets of simple substitutions. Note also that terms are allowed to occur only once as left hand sides in a substitution. However, it is possible that in some interpretation two different left hand sides have the same value. If in this case, the corresponding right hand sides do not have the same value, the substitution is called inconsistent. The notion of a consistent substitution will be formally defined in Chapter 4.

3.2.4 CONVENTION Substitutions are denoted by: \(\sigma, \sigma_0, \sigma_1, \ldots, \tau, \ldots\).

In the next definition the syntactic effect of a substitution \(\sigma\) on extended function symbols and terms is described. As substitutions may contain function updates, they may affect (extended) function symbols. The function updates in the substitution may also affect the (extended) terms which occur in the extended function symbol.

3.2.5 DEFINITION Let \(\sigma = [s_1 := t_1, \ldots, s_n := t_n]\) be a substitution. Then its effect \(f \sigma\) for \(f \in \text{XFUN}_{w,u}\) and \(t \in \text{XTERM}_u\) is mutually recursively defined by the following equations (where the corresponding cases of Definition 3.1.9 are considered in the same order):

\[
\begin{align*}
\perp \sigma &= \perp \\
x^u \sigma &= \begin{cases} 
t_i & \text{if } x^u \equiv s_i \text{ for some } i \ (1 \leq i \leq n), \\
x^u & \text{otherwise}, 
\end{cases} \\
c \sigma &= \begin{cases} 
t_i & \text{if } c \equiv s_i \text{ for some } i \ (1 \leq i \leq n), \\
c & \text{otherwise}, 
\end{cases} \\
g[\sigma] &= \begin{cases} 
g[r_{i_1} \mapsto t_{i_1}, \ldots, r_{i_k} \mapsto t_{i_k}] & \text{for all } [s_{i_j} := t_{i_j}] \in \sigma \\
& \text{such that } s_{i_j} \equiv g r_{i_j} \\
& (1 \leq i_1 < \ldots < i_k \leq n, \\
& 1 \leq j \leq k), \\
& \text{otherwise}, 
\end{cases} \\
g \sigma &= g \\
(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]) \sigma &= (g \sigma)[r_1 \sigma \mapsto s_1', \ldots, r_m \sigma \mapsto s_m'], \\
(g t) \sigma &= (g \sigma)(t \sigma).
\end{align*}
\]

If \(t = (t_1, \ldots, t_n) \in \text{XTERM}_w\), then \(t \sigma\) is defined by:

\[
t \sigma = (t_1 \sigma, \ldots, t_n \sigma).
\]

As substitution on terms is rather complicated, we have to verify that it is well-defined, i.e. that the application of substitutions to extended function symbols and extended terms does not produce objects of an unknown kind.
3.2. Substitution

3.2.6 Proposition Let a signature $\Sigma$ be given. Then the sets $\text{XFUN}_w,u$ and $\text{XTERM}_u$ are closed with respect to substitution for all $w \in \text{SORT}^+$ and all $u \in \text{SORT}$.

Proof By a simple, mutual induction on the complexity of the extended function symbols $f \in \text{XFUN}_w,u$ and extended terms $t \in \text{XTERM}$, using Definition 3.1.9. •

In order to define the effect of a substitution on formulae, we firstly define when a substitution affects a formula or a term. Roughly speaking, this is the case when this substitution syntactically changes this formula or term. It is, however, possible that a substitution is affecting without bringing about a syntactical change. For example, our definition will imply that the substitution $[x^u := x^v]$ affects the formula $\downarrow x^u$.

We also define the set of free variables of a substitution. These are the variables which might become part of extended terms or formulae after substitution. For example in the substitution $[f(x^u, g(x^v)) := z^u, y^v := z^v]$, the variables $x^u$, $x^v$ and $z^u$ are free, but the variable $y^v$ is not free.

3.2.7 Definition Let $\sigma = [s_1 := t_1, \ldots, s_m := t_m]$ be a substitution, and let $\varphi \in \text{FORM}$. Then $\sigma$ is said to affect $\varphi$, if for some $i$ $(1 \leq i \leq m)$:

- $s_i \equiv x^u$ and $x^u \in \text{FV}(\varphi)$, for some variable $x^u$, or
- $s_i \equiv c$ and $c$ occurs in $\varphi$, for some constant symbol $c$, or
- $s_i \equiv ft$ and $f$ occurs in $\varphi$, for some function symbol $f$ and vector $t$.

If $t \in \text{XTERM}$, then $\sigma$ is said to affect $t$ if the condition above holds with $\varphi$ replaced by $t$.

It is straightforward to recursively define when a constant or function symbol occurs in a term or formula.

3.2.8 Definition Let $\sigma = [s_1 := t_1, \ldots, s_m := t_m]$ be a substitution. Then the set $\text{FV}(\sigma)$ of free variables in $\sigma$ is defined by:

\[
\text{FV}(\sigma) = \bigcup_{i=1}^{m} \text{FV}(t_i) \cup \{\text{FV}(t) \mid s_i \equiv ft \text{ for some } i, 1 \leq i \leq m\}.
\]

3.2.9 Remark Before entering the next definition, we remind the reader to the fact that substitutions are sets of simple substitutions. As a consequence, if $\sigma$ and $\tau$ are substitutions, then $\sigma \setminus \tau$ denotes the difference of $\sigma$ and $\tau$ considered as sets, where set difference is defined in the usual way.
3.2.10 Definition Let \( \sigma = [s_1 := t_1, \ldots, s_m := t_m] \) be a substitution. Then its effect \( \varphi \sigma \) for \( \varphi \in \text{FORM} \) is recursively defined by:

\[
\begin{align*}
tt \sigma &= tt, \\
ff \sigma &= ff, \\
(\downarrow t) \sigma &= \downarrow(t \sigma), \\
(s = t) \sigma &= (s \sigma = t \sigma), \\
(Pt) \sigma &= P(t \sigma), \\
(\neg \psi) \sigma &= \neg(\psi \sigma), \\
(\psi \star \chi) \sigma &= (\psi \sigma \star \chi \sigma), \\
(Qx^u \psi) \sigma &= \begin{cases} 
Qx^u(\psi \sigma) & \text{if } \sigma \text{ does not affect } x^u, \\
Qy^u(\psi[x^u := y^u]\sigma) & \text{if } \sigma \text{ does not affect } x^u, \\
(Qx^u \psi) \sigma \backslash [s_i := t_i] & \text{if } x^u \equiv s_i \text{ for some } i \\
& (1 \leq i \leq m). 
\end{cases}
\]
\]

With respect to the second case of the last equation above, the variable \( y^u \) is called fresh if \( y^u \not\in \text{Fv}(\psi) \cup \text{Fv}(\sigma) \) and \( \sigma \) does not affect \( y^u \).

The notion of an affecting substitution is adequate in the following sense.

3.2.11 Proposition Let \( \sigma \) be a substitution, \( \varphi \in \text{FORM} \) and \( t \in \text{XTerm}_u \). If \( \sigma \) does not affect \( \varphi \) (or \( t \)), then \( \varphi \sigma \equiv \varphi \) (or \( \sigma \equiv t \)).

Proof The proof is by induction on the complexity of \( \varphi \) (or \( t \)). We will only consider the case that \( \varphi \equiv Qx^u \psi \), the other cases being elementary. So, suppose that \( \sigma \) does not affect \( Qx^u \psi \). We have three cases to consider:

- \( \sigma \) does not affect \( x^u \), and \( x^u \not\in \text{Fv}(\sigma) \) or \( \sigma \) does not affect \( \psi \).

Then \( (Qx^u \psi) \sigma = Qx^u(\psi \sigma) \) by definition. As \( \sigma \) does not affect \( Qx^u \psi \) and \( \sigma \) does not affect \( x^u \), \( \sigma \) cannot affect \( \psi \). Applying the induction hypothesis gives that \( \psi \sigma \equiv \psi \). This implies that \( \varphi \sigma \equiv \varphi \).

- \( \sigma \) does not affect \( x^u \), but \( x^u \in \text{Fv}(\sigma) \) and \( \sigma \) does affect \( \psi \).

This case is not possible. The symbols occurring in \( \psi \) are the same as those occurring in \( Qx^u \psi \), except that \( x^u \) might also freely occur in \( \psi \). So, if \( \sigma \) affects \( \psi \), but \( \sigma \) does not affect \( x^u \), there must be some other symbol than \( x^u \) in \( \psi \) which is affected by \( \sigma \). But this would imply that \( \sigma \) also affects \( Qx^u \psi \) which is excluded by assumption.
3.3. A Fitch-style deduction system for $E$-logic

- $\sigma$ does not affect $x^u$.

In this case $\sigma$ has to contain a simple substitution $[x^u := t]$. Now, $\sigma \setminus [x^u := t]$ does not affect $Qx^u \psi$, as already $\sigma$ does not affect this formula by assumption. Moreover, $\sigma \setminus [x^u := t]$ does not affect $x^u$. So, this case is reduced to one of the preceding two.

This settles these cases. ■

3.3 A Fitch-style deduction system for $E$-logic

The proof system presented in this section will be based on the system of natural deduction as proposed by Fitch in [Fit52]. A formalization of this system for classical predicate logic can be found in [WST93]. Our presentation will be different, however.

To begin with, a derivation of $\Gamma \vdash \varphi$ consists of a proof figure, and has the following form:

$$
\begin{array}{c}
\psi_1 \\
\vdots \\
\psi_n \\
\vdots \\
\varphi
\end{array}
$$

In this proof figure $\{\psi_1, \ldots, \psi_n\}$ is a finite subset of $\Gamma$. The idea behind this is that $\Gamma \vdash \varphi$ holds if $\varphi$ is derivable from a finite subset of $\Gamma$. The formulae in $\Gamma$ are called the assumptions of the derivation of $\varphi$.

If $n = 0$, then the derivation is a proof figure of the following format:

$$
\begin{array}{c}
\vdots \\
\varphi
\end{array}
$$

In both formats of a proof figure the dots above $\varphi$ denote formulae and/or proof figures. This means that proof figures are recursive structures. One might compare this recursive structures with the ones known from programming languages, e.g. the block structure in Algol 60.

The formulae above the horizontal rule of a proof figure are called the hypotheses of it. The order in which they appear is arbitrary. In most cases a proof figure has only one hypothesis. Only the outer proof figure of a derivation may have an arbitrary, but finite number of hypotheses. In our system, inner proof figures have at least one and at most two hypotheses. The ordered sequence of formulae below the horizontal rule is called the body of the proof figure.
Just like the notion of visible variables in programming languages, we have the notion of visible formulae. All hypotheses of a proof figure are visible in the body of it, and all formulae of it are visible in the proof figures below it that are contained in it.

Not all proof figures one can write down, are valid derivations. For any formula in a proof figure a condition must be met:

- It must either be a hypothesis (of a proof figure),
- or be an axiom,
- or be the result of applying a derivation rule to formulae and/or proof figures preceding it in the same proof figure.

In fact, the notion of derivation is recursively defined by the set of derivation rules. For the logical connectives we have introduction rules and elimination rules. An introduction rule for a connective describes how a new formula containing that connective can be created. Conversely, a corresponding elimination rule says how to use a formula containing that connective.

3.3.1 Definition A proof figure $\Pi$ is an ordered pair:

$$\Pi = \langle \Phi, \Psi \rangle,$$

where

- $\Phi$ is a finite set of formulae, called the hypotheses of $\Pi$,
- $\Psi$ is a finite list of formulae and/or proof figures, called the body of $\Pi$.

A formula $\varphi$ is contained in $\Pi$, if $\varphi \in \Phi$ or $\varphi \in \Psi$. A proof figure $\Pi'$ is contained in $\Pi$ if $\Pi' \in \Psi$. Notation: $\varphi \in \Pi$ and $\Pi' \in \Pi$, respectively.

3.3.2 Convention

i. Proof figures are denoted by: $\Pi, \Pi', \ldots, \Pi_1, \ldots$.

ii. Hypothesis sets of of proof figures are denoted by: $\Phi, \Phi', \ldots, \Phi_1, \ldots$.

iii. Bodies of proof figures are denoted by: $\Psi, \Psi', \ldots, \Psi_1, \ldots$.

3.3.3 Definition Let $\Pi = \langle \Phi, \Psi \rangle$ and $\Pi'$ be proof figures, and let $\varphi$ be a formula such that $\varphi \in \Pi$. Then $\varphi$ is called visible in $\Pi'$, if there exist a series of proof figures $\Pi_1, \ldots, \Pi_n$ ($n \geq 1$), such that:

- $\Pi' = \Pi_1 \in \ldots \in \Pi_n = \Pi$, and
- $\varphi \in \Phi$, or $\varphi \in \Psi$ and $\varphi$ precedes $\Pi_{n-1}$ in the list ordering of $\Psi$ (if $n \geq 2$).

Note that visibility is defined in such way that all hypotheses of a proof figure are visible in its body, and in the proof figures nested in it.
In making derivations, new nested proof figures are constructed. These nested proof figures introduce new hypotheses. Besides these hypotheses, all hypotheses of the proof figures in which the new one is nested, are ‘in power’ inside that new proof figure: they are active. Outside a proof figure the hypotheses of that proof figure are not active anymore: they have been discharged.

3.3.4 Definition Let a nested sequence \( \Pi_n = (\Phi_n, \Psi_n) \in \ldots \in \Pi_1 = (\Phi_1, \Psi_1) \) be given, then the set of active hypotheses of \( \Pi_n \) (with respect to \( \Pi_1 \)) is defined by \( \Phi_1 \cup \ldots \cup \Phi_n \).

So, the set of active hypotheses of a proof figure with respect to the derivation in which it is nested, is equal to the set of all hypotheses which are visible in the given proof figure, the hypotheses of that given proof figure included.

3.3.5 Definition Let \( \Phi \) be a finite set of formulae. Then, the proof figure \( \Pi = (\Phi, \Psi) \) is called a derivation from assumptions \( \Phi \) if any element of \( \Psi \) is generated using the following axioms and derivation rules. In formulating these we use the following conventions:

- \[ \frac{\Phi'}{\psi} \] denotes a proof figure with hypotheses \( \Phi' \), the last line of which contains the formula \( \psi \).

- A boxed formula denotes the conclusion of a rule.

- The formulae and/or proof figures above a conclusion denote the premises of a rule.

- In naming the rules, the following abbreviations are used: \( \text{I} \) and \( \text{E} \) denote introduction and elimination, respectively. Moreover, \( \text{l} \) and \( \text{r} \) denote left and right. The other names speak for themselves.

- \( \varphi, \psi, \chi \in \text{FORM}, s, t \in X\text{TERM}_u, f \in X\text{FUN}_{w,u}, P \in X\text{PRED}_w, (t_1, \ldots, t_n) \in X\text{TERM}^w, r_1, \ldots, r_m, t \in X\text{TERM}^w \) and \( s_1, \ldots, s_m \in X\text{TERM}_u \).

A derivation rule is called applicable if all premises of it are contained in the same proof figure. In that case the conclusion of that derivation rule may be appended to the body of the proof figure. Note that the hypotheses may appear in any order, but that the conclusion may not precede any hypothesis.

Any axiom may be inserted everywhere in the body of any proof figure in the derivation.

Anywhere in the body of a proof figure a new proof figure may be inserted, provided it will act as hypothesis for a derivation rule. In turn, the body of the new proof figure has to be generated by inserting axioms and/or applying derivation rules in it.
Axiom for \( \text{tt} \)
\[
\text{ttA: } \text{tt}
\]

Rules for \( \text{ff} \)
\[
\begin{align*}
\text{ffI: } & \varphi \\
\text{ffE: } & \text{ff} \\
& \varphi \\
& \text{ff}
\end{align*}
\]

Rules for negation
\[
\begin{align*}
\text{-I: } & \varphi \\
\text{-E: } & \text{ff} \\
& \varphi \\
& \text{ff}
\end{align*}
\]

Rules for conjunction
\[
\begin{align*}
\wedge\text{I: } & \varphi \\
\wedge\text{El: } & \varphi \land \psi \\
\wedge\text{Er: } & \varphi \land \psi \\
& \psi \\
& \varphi \land \psi
\end{align*}
\]

Rules for disjunction
\[
\begin{align*}
\lor\text{Ir: } & \varphi \\
\lor\text{II: } & \psi \\
\lor\text{E: } & \varphi \lor \psi \\
& \varphi \\
& \psi
\end{align*}
\]

Rules for implication
\[
\begin{align*}
\rightarrow\text{I: } & \varphi \\
\rightarrow\text{E: } & \varphi \rightarrow \psi \\
& \psi \\
& \varphi \rightarrow \psi
\end{align*}
\]

Rules for equivalence
\[
\begin{align*}
\leftrightarrow\text{I: } & \varphi \leftrightarrow \psi \\
\leftrightarrow\text{El: } & \varphi \leftrightarrow \psi \\
\leftrightarrow\text{Er: } & \varphi \leftrightarrow \psi \\
& \varphi \\
& \psi
\end{align*}
\]
3.3. A Fitch-style deduction system for E-logic

Rules for the universal quantifier

\[ \forall I: \quad \frac{\varphi[x^u:=y^u]}{\forall x^u \varphi} \quad \forall E: \quad \frac{\varphi[x^u:=t]}{\varphi[x^u:=t]} \]

Condition for application of \( \forall I \): \( y^u \notin Fv(\varphi) \) or \( y^u \equiv x^u \), and \( y^u \) does not freely occur in the active hypotheses (of the proof figure in which \( \forall I \) is applied, and with respect to the derivation).

Rules for the existential quantifier

\[ \exists I: \quad \frac{\varphi[x^u:=t]}{\exists x^u \varphi} \quad \exists E: \quad \frac{\varphi[x^u:=y^u]}{\psi} \]

Condition for application of \( \exists E \): \( y^u \notin Fv(\varphi) \) or \( y^u \equiv x^u \), and \( y^u \notin Fv(\psi) \), and \( y^u \) may not freely occur in the active hypotheses.

Axiom for \( \bot \)

\[ \bot A: \quad \neg \bot \]

Rules for equality

\[ = I: \quad \frac{\bot}{t = t} \quad \frac{\bot}{t = t} \quad \frac{\bot}{t = \varphi[x^u:=s]} \quad \frac{\bot}{t = \varphi[x^u:=t]} \]

Rules for strictness

\[ f Str: \quad \frac{\bot}{f(t_1, \ldots, t_i, \ldots, t_n)} \quad \frac{\bot}{f(t_i)} \quad \frac{\bot}{P(t_1, \ldots, t_i, \ldots, t_n)} \]

Rules for extended function symbols

In the following rules the following abbreviations have been adopted:

- If \( t = (t_1, \ldots, t_n) \), then \( \bot t \) means:

\[ \bot t_1 \land \ldots \land \bot t_n. \]
• If \( r = (r_1, \ldots, r_n) \) and \( t = (t_1, \ldots, t_n) \), then \( r = t \) abbreviates:

\[
 r_1 = t_1 \land \ldots \land r_n = t_n.
\]

• The proof figure in rule \textbf{Xf1} which acts as hypothesis, represents a series of \( m - 1 \) proof figures: one proof figure for each value of \( j \) in the range \( 1, \ldots, m \) such that \( j \neq i \).

\textbf{Xf1}: \( t = r_i \)
\[\downarrow s_i\]
\[
\begin{array}{c}
 t = r_j \\
 s_i = s_j
\end{array}
\]
\[
[f[r_1 \mapsto s_1, \ldots, r_i \mapsto s_i, \ldots, r_m \mapsto s_m]t = s_i]
\]

\textbf{Xf2}: \( t \neq r_1 \)
\[\vdots\]
\( t \neq r_m \)
\[\downarrow ft\]
\[
[f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = ft]
\]

\textbf{Xf3}: \( t = r_i \)

\[\neg \downarrow s_i\]

\[\neg \downarrow [\ldots, r_i \mapsto s_i, \ldots]t\]

\textbf{Xf4}: \( t = r_i \)
\( t = r_j \)
\( s_i \neq s_j \)

\[\neg \downarrow [\ldots, r_i \mapsto s_i, \ldots, r_j \mapsto s_j, \ldots]t \quad (i \neq j)\]

\textbf{Xf5}: \( t \neq r_1 \)
\[\vdots\]
\( t \neq r_m \)

\[\neg \downarrow ft\]

\[\neg \downarrow [r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t\]

\textbf{Reiteration rule}

\textbf{Rei:} \[
\phi
\]

\textit{Condition:} \( \phi \) has to be visible.
3.3. A Fitch-style deduction system for E-logic

3.3.6 REMARK In the version of E-logic described in [TD88] a rule called sub is used, which, under certain conditions, allows to instantiate free variables by terms. Since we introduced a special term $\bot$ to denote the 'undefined value', we do not need this rule. In fact, the formula

$$\forall x^u \varphi \land \varphi[x^u := \bot]$$

expresses that $\varphi$ holds for any term substituted for $x^u$, even if it is non-denoting. As a result, in our logic free variables are not schematic, as in [TD88], which simplifies the semantics of free variables.

3.3.7 REMARK If we leave out rule $\neg$E, we have an intuitionistic variant of E-logic.

3.3.8 DEFINITION Let $\varphi \in \text{FORM}$ and $\Gamma \subseteq \text{FORM}$. Then $\varphi$ is derivable from $\Gamma$, notation $\Gamma \vdash \varphi$, if there exist a derivation $\Pi = (\Phi, \Psi)$ such that $\Phi \subseteq \Gamma$, and $\varphi \in \Psi$.

Regarding the preceding definition, if $\Pi = (\Phi, \Psi)$ is a derivation of $\Gamma \vdash \varphi$, then $\varphi$ normally is the last line of it, i.e. it is the last element of the list $\Psi$. If $\Gamma = \emptyset$, then, of course, $\Phi = \emptyset$.

3.3.9 PROPOSITION The equality relation $=$ is strict, i.e. $\vdash s = t \rightarrow \downarrow s \land \downarrow t$ for any $s, t \in \text{XTERM}_u$ and $u \in \text{SORT}$.

PROOF The derivation of the strictness of $=$ can be found in Figure 3.1.

3.3.10 REMARK From the derivation in Figure 3.1 it follows that equality is symmetric (see lines 1 up to 13). Furthermore, it is easy to see that equality is transitive. However, it does not follow that $=$ is an equivalence relation, since $t = t$ only holds if $\downarrow t$.

The next proposition shows how definedness can be defined in terms of the existential quantifier and equality.

3.3.11 PROPOSITION Let $t \in \text{XTERM}_u$, then

$$\vdash \downarrow t \leftrightarrow \exists x^u[x^u = t].$$

PROOF The derivation is shown in Figure 3.2.

The notion of equality the axioms deal with, is called weak equality. The stronger notion $s \approx t$, called strong equality, holds if $s$ and $t$ are both defined and equal, or if $s$ and $t$ are both undefined. It can be defined in terms of $\downarrow$ and $=$ as follows.

3.3.12 DEFINITION Let $s, t \in \text{XTERM}_u$, then strong equality, denoted by $\approx$, is defined by:

$$s \approx t \equiv (\downarrow s \lor \downarrow t \rightarrow s = t).$$
Chapter 3. The syntax of E-logic

1. \[ s = t \] (Hyp.)
2. \[ t = t \] (Sub, 1, 1)
3. \[ \downarrow t \] (=E, 2)
4. \[ \downarrow s \lor \downarrow s \] (Hyp.)
5. \[ \downarrow s \] (Hyp.)
6. \[ s = s \] (=I, 5)
7. \[ \downarrow s \] (Hyp.)
8. \[ s = s \] (=I, 7)
9. \[ s = s \] (∨E, 4, 5, 6, 7, 8)
10. \[ \downarrow s \lor \downarrow s \rightarrow s = s \] (→I, 4, 9)
11. \[ \downarrow t \lor \downarrow s \rightarrow t = s \] (Sub, 1, 10)
12. \[ \downarrow t \lor \downarrow s \] (∨Ir, 3)
13. \[ t = s \] (→E, 11, 12)
14. \[ s = s \] (Sub, 13, 13)
15. \[ \downarrow s \] (=E, 14)
16. \[ \downarrow s \land \downarrow t \] (∧I, 1, 15)
17. \[ s = t \rightarrow \downarrow s \land \downarrow t \] (→I, 1, 16)

Figure 3.1: \( \vdash s = t \rightarrow \downarrow s \land \downarrow t \).

1. \[ \downarrow t \] (Hyp.)
2. \[ t = t \] (=I, 1)
3. \[ \exists x^u[x^u = t] \] (=I, 1, 2)
4. \[ \downarrow t \rightarrow \exists x^u[x^u = t] \] (→I, 1, 3)
5. \[ \exists x^u[x^u = t] \] (Hyp.)
6. \[ \downarrow y^u \] (Hyp.)
7. \[ y^u = t \] (Hyp.)
8. \[ \downarrow t \] (Sub, 6, 7)
9. \[ \downarrow t \] (∃E, 5, 6, 7, 8)
10. \[ \exists x^u[x^u = t] \rightarrow \downarrow t \] (→I, 5, 9)
11. \[ \downarrow t \leftrightarrow \exists x^u[x^u = t] \] (↔I, 4, 10)

Figure 3.2: \( \vdash \downarrow t \leftrightarrow \exists x^u[x^u = t] \).
3.3. A Fitch-style deduction system for $E$-logic

1. $\neg \downarrow t$ (Hyp.)
2. $\downarrow t \lor \downarrow \bot$ (Hyp.)
3. $\downarrow t$ (Hyp.)
4. $\neg \downarrow t$ (Rei, 1)
5. $\text{ff}$ (ffI, 3, 4)
6. $\downarrow \bot$ (Hyp.)
7. $\neg \downarrow \bot$ (⊥A)
8. $\text{ff}$ (ffI, 6, 7)
9. $\text{ff}$ ($\lor$E, 2, 3, 5, 6, 8)
10. $t = \bot$ (ffE, 9)
11. $t \approx \bot$ ($\rightarrow$I, 2, 10)
12. $\neg \downarrow t \rightarrow t \approx \bot$ ($\rightarrow$I, 1, 11)
13. $t \approx \bot$ (Hyp.)
14. $\downarrow t$ (Hyp.)
15. $\downarrow t \lor \downarrow \bot$ ($\lor$I, 14)
16. $t \approx \bot$ (Rei, 13)
17. $t = \bot$ ($\rightarrow$E, 15, 16)
18. $\downarrow \bot$ (from 17, by strictness $\Rightarrow$)
19. $\neg \downarrow \bot$ (⊥A)
20. $\text{ff}$ (ffI, 18, 19)
21. $\neg \downarrow t$ ($\neg$I, 14, 20)
22. $t \approx \bot \rightarrow \neg \downarrow t$ ($\rightarrow$I, 13, 21)
23. $\neg \downarrow t \leftrightarrow t \approx \bot$ ($\leftrightarrow$I, 12, 22)

Figure 3.3: $\vdash \neg \downarrow t \leftrightarrow t \approx \bot$.

3.3.13 Remark Formally, $s \approx t$ is an abbreviation of $\downarrow s \lor \downarrow t \rightarrow s = t$.

The next proposition deals with the relation between undefinedness and $\bot$.

3.3.14 Proposition Let $t \in X_{\text{TERM}_u}$, then:

$\vdash \neg \downarrow t \leftrightarrow t \approx \bot$.

Proof The derivation is shown in Figure 3.3. \[ \blacksquare \]

3.3.15 Convention If $r = (r_1, \ldots, r_n)$ and $t = (t_1, \ldots, t_n)$, then $r \approx t$ abbreviates:

$r_1 \approx t_1 \land \ldots \land r_n \approx t_n$. 

Figure 3.4: \( \vdash s \approx t \rightarrow t \approx s \).

3.3.16 Theorem The relation \( \approx \) on (extended) terms is a congruence relation, i.e. \( \approx \) is an equivalence relation, and:

\[ \vdash s \approx t \rightarrow r[x^u:=s] \approx r[x^u:=t] \]

for any \( r \in \text{XTERM}_v \) and \( s, t \in \text{XTERM}_u \).

Proof The proof that \( \approx \) is symmetric, can be found in Figure 3.4. Reflexivity and transitivity are left to the reader.

The proof of the second property is by mutual induction on the structure of extended terms \( r \) and extended function symbols \( f \).

- \( r \equiv \bot, r \equiv x^u, r \equiv x^v \) where \( x^u \neq x^v \), or \( r \equiv c \).
  
  In this case the derivation is easy to find.

- \( f \in \text{FUN}_{w,v} \) and \( t \in \text{XTERM}_w \).
  
  This is the base case for extended functions.

Firstly, observe that by Definition 3.2.5 \( f[x^u:=s] \equiv f \equiv f[x^u:=t] \). From the reflexivity of \( \approx \) now immediately follows:

\[ \vdash s \approx t \rightarrow f[x^u:=s]t \approx f[x^u:=t]t, \]

which solves the base case.
3.3. A Fitch-style deduction system for \(E\)-logic

- \(f \equiv g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]\), where \(g \in X\text{FUN}_{u,v}\), \(r_1, \ldots, r_m \in X\text{TERM}^u\), and \(s_1, \ldots, s_m \in X\text{TERM}^v\).

This is the induction step for extended functions, the most complicated case. We have to prove that for any \(t \in X\text{TERM}^u\):

\[\vdash s \approx t \rightarrow f[x^u:=s]t \approx f[x^u:=t]t.\]

Firstly observe, that by Definition 3.2.5:

\[\begin{align*}
(g[\ldots, r_i \mapsto s_i, \ldots])[x^u:=s] & \equiv (g[x^u:=s])[\ldots, r_i[x^u:=s] \mapsto s_i[x^u:=s], \ldots], \\
(g[\ldots, r_i \mapsto s_i, \ldots])[x^u:=t] & \equiv (g[x^u:=t])[\ldots, r_i[x^u:=t] \mapsto s_i[x^u:=t], \ldots].
\end{align*}\]

By the induction hypotheses there exist the following derivations:

\[\Pi_1 : \vdash s \approx t \rightarrow g[x^u:=s]t \approx g[x^u:=t]t,\]
\[\Pi_{2i} : \vdash s \approx t \rightarrow r_i[x^u:=s] \approx r_i[x^u:=t],\]
\[\Pi_{3i} : \vdash s \approx t \rightarrow s_i[x^u:=s] \approx s_i[x^u:=t].\]

The fact that these derivations exist will be used in the 'informal' derivation in Figures 3.5 and 3.6. In this derivation we will assume (without loss of generality) that \(i\) ranges over \(\{1, 2\}\).

Furthermore, we will use the following abbreviations. For any expression \(expr\) we will write \(expr[s]\) instead of \(expr[x^u:=s]\). Moreover, instead of \(g[\ldots, r_i[x^u:=s] \mapsto s_i[x^u:=s], \ldots]\) we will write \(g[r_i[s] \mapsto s_i[s]]\). Lastly, some obvious intermediate derivation steps (such applications of reiteration), as well as line numbers in annotations will be omitted.

- \(r \equiv fr\), where \(f \in X\text{FUN}_{u,v}\) and \(r \in X\text{TERM}^u\).

By Definition 3.2.5 we have:

\[\begin{align*}
(fr)[x^u:=s] & \equiv (f[x^u:=s])(r[x^u:=s]), \\
(fr)[x^u:=t] & \equiv (f[x^u:=t])(r[x^u:=t]).
\end{align*}\]

By the induction hypotheses there exist the following derivations:

\[\Pi_1 : \vdash s \approx t \rightarrow f[x^u:=s]t \approx f[x^u:=t]t,\]
\[\Pi_2 : \vdash s \approx t \rightarrow r[x^u:=s] \approx r[x^u:=t].\]
\[ s \approx t \]

\[ \downarrow g[s][r_1[s] \rightarrow s_1[s]]t \vee \downarrow g[t][r_1[t] \rightarrow s_1[t]]t \]

\[ \downarrow g[s][r_1[s] \rightarrow s_1[s]]t \]

\[ t = r_1[s] \vee t = r_2[s] \vee (t \neq r_1[s] \wedge t \neq r_2[s]) \]

\[ t = r_1[s] \]

\[ \neg \downarrow s_1[s] \]

\[ \neg \downarrow g[s][r_1[s] \rightarrow s_1[s]]t \]

\[ \downarrow s_1[s] \]

\[ t = r_2[s] \]

\[ s_1[s] \neq s_2[s] \]

\[ \neg \downarrow g[s][r_1[s] \rightarrow s_1[s]]t \]

\[ s_1[s] = s_2[s] \]

\[ g[s][r_1[s] \rightarrow s_1[s]]t = s_1[s] \]

\[ s_1[s] = s_1[t] \]

\[ t = r_2[s] \rightarrow s_1[s] = s_2[s] \]

\[ \downarrow r_1[s] \]

\[ r_1[s] = r_1[t] \]

\[ t = r_1[t] \]

\[ \downarrow s_1[t] \]

\[ t = r_2[t] \]

\[ \downarrow r_2[t] \]

\[ r_2[s] = r_2[t] \]

\[ t = r_2[s] \]

\[ s_1[s] = s_2[s] \]

\[ \downarrow s_2[s] \]

\[ s_2[s] = s_2[t] \]

\[ s_1[t] = s_2[t] \]

\[ g[t][r_1[t] \rightarrow s_1[t]]t = s_1[t] \]

\[ g[s][r_1[s] \rightarrow s_1[s]]t = g[t][r_1[t] \rightarrow s_1[t]]t \]

\[ t = r_2[s] \]

\[ \vdots \]

\[ g[s][r_1[s] \rightarrow s_1[s]]t = g[t][r_1[t] \rightarrow s_1[t]]t \]

\[ (\text{Hyp.}) \]

\[ (\text{Hyp.}) \]

\[ (\text{Hyp.}) \]

\[ (\text{classical}) \]

\[ (\text{Hyp.}) \]

\[ (\text{Hyp.}) \]

\[ (\text{Xf3}) \]

\[ (\neg \text{I}, \neg \text{E}) \]

\[ (\text{Hyp.}) \]

\[ (\text{Hyp.}) \]

\[ (\text{Xf4}) \]

\[ (\neg \text{I}, \neg \text{E}) \]

\[ (\text{Xf1}) \]

\[ (\forall \text{Ir}, \Pi_{31}, \rightarrow \text{E}) \]

\[ (\rightarrow \text{I}) \]

\[ (\text{strictness} =) \]

\[ (\forall \text{Ir}, \Pi_{21}, \rightarrow \text{E}) \]

\[ (\text{Sub}) \]

\[ (\text{strictness} =) \]

\[ (\text{Hyp.}) \]

\[ (\text{strictness} =) \]

\[ (\forall \Pi_{12}, \rightarrow \text{E}) \]

\[ (\text{symmetry}, \text{Sub}) \]

\[ (\text{strictness} =) \]

\[ (\forall \text{Ir}, \Pi_{32}, \rightarrow \text{E}) \]

\[ (\text{Sub}) \]

\[ (\text{Xf1}) \]

\[ (\text{symmetry}, \text{Sub}) \]

\[ (\text{Hyp.}) \]

\[ (\text{analogously}) \]

Figure 3.5: \( \vdash s \approx t \rightarrow g[r_1 \mapsto s_1][x^u := s]t \approx g[r_1 \mapsto s_1][x^u := t]t \) (part 1).
3.3. A Fitch-style deduction system for E-logic

\[ \begin{array}{ll}
\text{t} \neq r_1[s] \land t \neq r_2[s] & \text{\textbf{(Hyp.)}} \\
\Box \text{t} \neq r_1[s] & \text{\textbf{(\AE I)}} \\
\Box \text{t} \neq r_2[s] & \text{\textbf{(\AE E)}} \\
\text{t} = r_1[t] & \text{\textbf{(Hyp.)}} \\
\downarrow r_1[t] & \text{\textbf{(strictness \(=\))}} \\
r_1[s] = r_1[t] & \text{\textbf{(\forall I, \Pi 21, \rightarrow E)}} \\
\text{t} \neq r_1[s] & \text{\textbf{(\AE)}} \\
\text{t} \neq r_1[t] & \text{\textbf{(Sub)}} \\
\text{t} \neq r_1[t] & \text{\textbf{(ffI, \neg I)}} \\
\vdots & \\
\text{t} \neq r_2[t] & \text{\textbf{(analogously)}} \\
\downarrow g[s] & \text{\textbf{(Hyp.)}} \\
\downarrow g[s][r_1[s] \rightarrow s_1[s]] t & \text{\textbf{(Xf5)}} \\
\downarrow g[s] t & \text{\textbf{(ffI, \neg I, \neg E)}} \\
g[s] t = g[t] t & \text{\textbf{(\forall I, \Pi 1, \rightarrow E)}} \\
\downarrow g[t] t & \text{\textbf{(strictness \(=\))}} \\
g[s][r_1[s] \rightarrow s_1[s]] t = g[s] t & \text{\textbf{(Xf2)}} \\
g[t][r_1[t] \rightarrow s_1[t]] t = g[t] t & \text{\textbf{(Xf2)}} \\
g[s][r_1[s] \rightarrow s_1[s]] t = g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(symmetry, Sub)}} \\
g[s][r_1[s] \rightarrow s_1[s]] t = g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(\forall E)}} \\
\downarrow g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(Hyp.)}} \\
\vdots & \\
g[s][r_1[s] \rightarrow s_1[s]] t = g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(analogously)}} \\
g[s][r_1[s] \rightarrow s_1[s]] t = g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(\forall E)}} \\
g[s][r_1[s] \rightarrow s_1[s]] t \approx g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(\rightarrow I)}} \\
s \approx t \rightarrow g[s][r_1[s] \rightarrow s_1[s]] t \approx g[t][r_1[t] \rightarrow s_1[t]] t & \text{\textbf{(part 2).}} \\
\end{array} \]

Figure 3.6: \( \vdash s \approx t \rightarrow g[r_1 \rightarrow s_1][x^u := s] t \approx g[r_1 \rightarrow s_1][x^u := t] t \) (part 2).
where $t \in \text{XTERM}_w$ is arbitrary. In the derivation, which is shown in Figure 3.7, we will use the same abbreviations and conventions as in the previous case.

This finishes the proof of Theorem 3.3.16.

Note that the derivation in Figures 3.5 and 3.6 is not constructive, i.e. the rule $\neg \Phi$ is used. This is done several times: in the step marked ‘classical’, and in the steps where this is explicitly mentioned.

3.3.17 Theorem Let $\varphi \in \text{FORM}$ and $s, t \in \text{XTERM}_u$. Then the following substitution rule is derivable:

\begin{align*}
\approx \text{Sub}: \\
\varphi[x^u:=s] \\
s \approx t \\
\varphi[x^u:=t]
\end{align*}

Proof The proof is on the complexity of $\varphi$, using the strictness property of predicates, $\approx$, and $\lfloor$, and the fact that $\approx$ is a congruence relation.
Chapter 4

The semantics of $E$-logic

As we are considering a classical version of $E$-logic in this thesis, we will formulate a Tarski style semantics for it. Like structures for so-called free logic, structures for $E$-logic may have empty domains. Moreover, functions on these structures are allowed to be partial. These so-called partial structures, which we will define in Section 4.1, are straightforward generalizations of partial algebras (see [Rei87]) which deal with partial functions. In partial structures also predicates are interpreted. Their denotation will not be partial, however.

Section 4.2 deals with operations on interpretations. One of these operations, the modification operator, changes the interpretation of function symbols. It will be used to formulate the substitution theorem which is valid for general substitutions (modulo some consistency condition). The other two operations are used to extend or contract universes. They play a role in connection with the extension and contraction updates of evolving algebras.

Then, in Section 4.1 we will show that our system of natural deduction is both sound and complete. To prove completeness we will use the Lindenbaum-Henkin construction. The completeness result holds for arbitrary formulae, i.e. formulae containing extended function symbols.

Finally, in Section 4.4, we will introduce standard $E$-logic. In this kind of logic a fixed part of a signature gets assigned a special interpretation.

4.1 Partial structures and interpretations

4.1.1 Convention i. We recall Convention 3.1.7 were the notation $\text{TERM}^w$ has been introduced. Now, let a family $\{A_u \mid u \in \text{SORT}\}$ of sets be given. Then for any $w \equiv u_1 \ldots u_n \in \text{SORT}^*$ the set $A^w$ is defined by $A^w = A_{u_1} \times \ldots \times A_{u_n}$. By convention $A^\emptyset = \{\emptyset\}$.

ii. The notation $f : A \rightarrow B$ will be used to indicate that $f$ is a possibly partial function from $A$ to $B$. If $f : A \rightarrow B$, then $\text{DOM}(f)$ denotes the set of all elements $x \in A$ for which $f(x)$ is defined. Clearly, $\text{DOM}(f) = A$ whenever $f$ is a total function, and $\text{DOM}(f) \subseteq A$, otherwise. Note that total functions are considered to be special partial functions.
In specifying a partial function \( f \), we will define its domain \( \text{DOM}(f) \) and the values \( f(x) \) it takes over this domain. Of course, the expression \( f(x) \) only makes sense for \( x \in \text{DOM}(f) \), so this will tacitly be assumed in the specification.

iii. In the metalanguage we will use the symbols = and \( \in \). Their interpretation will be existential. So \( s = t \) holds for some terms \( s \) and \( t \) in the metalanguage if both \( s \) and \( t \) are defined values which are equal. Furthermore \( s \in t \) holds if \( s \) and \( t \) are both defined values which stand in the intended membership relation.

4.1.2 Definition Let \( \Sigma = \langle \Omega, \text{SORT}, \text{SYM}, \sigma \rangle \) be a given signature. A many-sorted partial structure \( \mathfrak{A} \) over \( \Sigma \) is an ordered pair:

\[
\mathfrak{A} = \langle A, F \rangle,
\]

such that:

1. \( A \) is a \text{SORT}-indexed family of, possibly empty, sets, i.e.

\[
A = \{ \mathfrak{A}_u \mid u \in \text{SORT} \}.
\]

The elements of \( A \) are called the universes of \( \mathfrak{A} \).

2. \( F \) is a \( \sigma \)-indexed family of partial functions on \( A \), i.e.

\[
F = \{ f^\mathfrak{A} \mid f \in \text{SYM} \},
\]

where

- \( f^\mathfrak{A} : \mathfrak{A}^w \rightarrow \mathfrak{A}_u \) whenever \( \sigma(f) = (w, u) \) and \( u \in \text{SORT} \),
- \( f^\mathfrak{A} : \mathfrak{A}^w \rightarrow \mathfrak{B} \) whenever \( \sigma(f) = (w, \Omega) \).

A many-sorted partial structure over \( \Sigma \) will be called a \( \Sigma \)-structure.

4.1.3 Remark i. In the definition above, if \( \sigma(P) = (w, \Omega) \), then \( P^\mathfrak{A} \) is a \textit{total} function with codomain \( \mathfrak{B} \), the set of booleans. In other words, \( P^\mathfrak{A} \) is the characteristic function of the predicate symbol \( P \). We will treat \( P^\mathfrak{A} \) not as a function, however, but as the corresponding set.

ii. If \( \sigma(c) = (\lambda, u) \), then \( c^\mathfrak{A} \) is a partial function \( c^\mathfrak{A} : \{ \emptyset \} \rightarrow \mathfrak{A}_u \). This means that the (set theoretical) graph of \( c^\mathfrak{A} \) is either the set \( \{(\emptyset, d)\} \) for some element \( d \in \mathfrak{A}_u \), or the empty set \( \emptyset \). In the first case \( c \) is denoting the element \( d \), in the latter case \( c \) is uninterpreted by \( \mathfrak{A} \).

4.1.4 Convention \( \Sigma \)-structures are denoted by \( \mathfrak{A}, \mathfrak{A}_0, \mathfrak{A}_1, \ldots, \mathfrak{B}, \ldots \).

We will now explain how a many-sorted partial structure over \( \Sigma \) induces an interpretation of terms and formulae over \( \Sigma \). In order to deal with free variables in terms and formulae the notion of a \textit{(variable) assignment} is needed.
4.1.5 Definition Let $\mathfrak{A}$ be a $\Sigma$-structure, and let $\{\text{VAR}_u \mid u \in \text{SORT}\}$ be the family of sets of variables associated with $\Sigma$. A (variable) assignment $\beta$ (w.r.t. $\Sigma$ and $\mathfrak{A}$) is a SORT-indexed family of partial functions $\{\beta^u \mid u \in \text{SORT}\}$ such that $\beta^u : \text{VAR}_u \rightarrow \mathfrak{A}_u$ for all $u \in \text{SORT}$.

Note that variable assignments are partial functions. This means that variables are not necessarily denoting terms.

4.1.6 Convention
i. The following symbols are used to denote variable assignments: $\beta, \beta_0, \beta_1, \ldots, \gamma, \ldots$.

ii. If $\beta$ is a variable assignment, we will write $\beta(x^u)$ instead of $\beta^u(x^u)$ ($x^u \in \text{VAR}_u, u \in \text{SORT}$).

4.1.7 Definition A $\Sigma$-interpretation for a language with signature $\Sigma$ is an ordered pair $\langle \mathfrak{A}, \beta \rangle$ where $\mathfrak{A}$ is a $\Sigma$-structure and $\beta$ is a variable assignment.

By means of an interpretation terms and formulae can be interpreted. Due to the fact that we deal with a logic of partial functions, the interpretation of terms will be partial. This means that not all terms will be denoting. The interpretation of formulae will, however, be total. Any formula will receive either the truth value true, or false.

In order to facilitate reasoning about non-denoting terms in the metalanguage we will make use of the notion of a lifted set, where we do not take into consideration any ordering on this set. A lifted set contains a special element $\infty$ which denotes the 'value' undefined. By means of lifted sets partial functions can be transformed into total functions in the standard way. Note that according to Convention 4.1.1(iii) the equality $\infty = \infty$ holds: $\infty$ is denoting some value in the metalanguage.

4.1.8 Definition Let $V$ be some set, then $V_\infty$, the lifting of $V$, is defined as $V \cup \{\infty_V\}$ for some element $\infty_V \notin V$.

4.1.9 Convention We will always suppress the subscript $V$ in $\infty_V$ for any set $V$.

4.1.10 Definition Let a signature $\Sigma$ and a $\Sigma$-interpretation $\langle \mathfrak{A}, \beta \rangle$ be given. The denotation $t^{\mathfrak{A},\beta}$ of a term $t \in \text{TERM}_u$ is such that $t^{\mathfrak{A},\beta} \in (\mathfrak{A}_u)_\infty$ for all $u \in \text{SORT}$, and is recursively defined by the equations below, where the notation $t^{\mathfrak{A},\beta}$ for a vector $t = (t_1, \ldots, t_n)$ refers to $(t_1^{\mathfrak{A},\beta}, \ldots, t_n^{\mathfrak{A},\beta})$.

$$\perp^{\mathfrak{A},\beta} = \infty,$$

$$x^{u^{\mathfrak{A},\beta}} = \begin{cases} \beta(x^u) & \text{if } x^u \in \text{DOM}(\beta^u), \\
\infty & \text{otherwise}, \end{cases}$$

$$c^{\mathfrak{A},\beta} = \begin{cases} d & \text{if } c^{\mathfrak{A}} = \{(\emptyset, d)\}, \\
\infty & \text{otherwise}, \end{cases}$$
\[(f t)^{\alpha, \beta} = \begin{cases} f^\alpha t^\alpha, & \text{if } t^\alpha, \beta \in \text{DOM}(f^\alpha), \\ \infty, & \text{otherwise.} \end{cases} \]

In order to deal with the denotation of extended function symbols we employ an auxiliary condition \(C(i)\) on these symbols in the next definition. If \(f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \in \text{XFUN}_{w,u}\), then \(C(i)\) expresses that all components of the vector \(r_i\), as well as \(s_i\) are denoting, and that if for different \(i\) and \(j\) the denotations of \(r_i\) and \(r_j\) are the same, then also the denotations of \(s_i\) and \(s_j\) are the same (\(1 \leq i, j \leq m\)).

As one might expect, the denotations of extended function symbols and extended terms are defined by mutual recursion. Moreover, Convention 4.1.1 is used when defining (partial) functions.

4.1.11 Definition Let a signature \(\Sigma\) and a \(\Sigma\)-interpretation \((\mathfrak{A}, \beta)\) be given. The denotation \(f^{\alpha, \beta}\) of an extended function symbol \(f \in \text{XFUN}_{w,u}\) and the denotation \(t^{\alpha, \beta}\) of an extended term \(t \in \text{XTERM}_u\) is such that \(f^{\alpha, \beta}: \mathfrak{A}^w \to \mathfrak{A}_u\) and \(t^{\alpha, \beta} \in (\mathfrak{A}_u)_\infty\) for all \(w \in \text{SORT}^+\) and \(u \in \text{SORT}\).

Let \(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \in \text{XFUN}_{w,u}\) and \(1 \leq i \leq m\), then \(C(i)\) will be used as an abbreviation for:

\[
\begin{align*}
& r_i^{\alpha, \beta} \in \mathfrak{A}^w \quad & \& \quad s_i^{\alpha, \beta} \in \mathfrak{A}_u \\
& \quad & \& \quad & \& \quad & \begin{array}{c}
\quad \forall j \cdot [1 \leq j \leq m \quad & \& \quad i \neq j \quad & \& \quad r_j^{\alpha, \beta} = r_j^{\alpha, \beta} \Rightarrow s_i^{\alpha, \beta} = s_j^{\alpha, \beta}].
\end{array}
\end{align*}
\]

Now, if \(f = g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \in \text{XFUN}_{w,u}\), then \(f^{\alpha, \beta}\) is recursively defined by:

\[
\begin{align*}
\text{DOM}(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\alpha, \beta}) \\
= \text{DOM}(g^{\alpha, \beta}) \setminus \{r_i^{\alpha, \beta} \mid \sim C(i) \quad & \& \quad 1 \leq i \leq m\} \\
\cup \{r_i^{\alpha, \beta} \mid C(i) \quad & \& \quad 1 \leq i \leq m\}
\end{align*}
\]

and

\[
g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\alpha, \beta} x = \begin{cases} s_i^{\alpha, \beta}, & \text{if } x = r_i^{\alpha, \beta}, \\
\vdots & \\
s_m^{\alpha, \beta}, & \text{if } x = r_m^{\alpha, \beta}, \\
g^{\alpha, \beta} x, & \text{otherwise.} \end{cases}
\]

In addition, if \(g \in \text{FUN}_{w,u}\), then:

\[
g^{\alpha, \beta} = g^\alpha.
\]

Furthermore, if \(t \in \text{XTERM}_u\), then \(t^{\alpha, \beta}\) is defined by:

\[
\bot^{\alpha, \beta} = \infty,
\]

\[
(x^u)^{\alpha, \beta} = \begin{cases} \beta(x^u), & \text{if } x^u \in \text{DOM}(\beta^u), \\
\infty & \text{otherwise,} \end{cases}
\]

Chapter 4. The semantics of E-logic
4.1. Partial structures and interpretations

\[ c^{\alpha, \beta} = \begin{cases} \ d & \text{if } c^{\alpha} = \{ (\emptyset, d) \}, \\ \infty & \text{otherwise}, \end{cases} \]

\[ (gt)^{\alpha, \beta} = \begin{cases} \ g^{\alpha, \beta} t^{\alpha, \beta} & \text{if } t^{\alpha, \beta} \in \text{DOM}(g^{\alpha, \beta}), \\ \infty & \text{otherwise}. \end{cases} \]

4.1.12 REMARK In the recursive definition of \( g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\alpha, \beta} \) above, we have stipulated that \( g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\alpha, \beta} \mathbf{x} = g^{\alpha, \beta} \mathbf{x} \) whenever \( \mathbf{x} \in \text{DOM}(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\alpha, \beta}) \) and \( \mathbf{x} \neq r_i^{\alpha, \beta} \) for all \( i \) such that \( 1 \leq i \leq m \). However, this is only meaningful if \( \mathbf{x} \in \text{DOM}(g^{\alpha, \beta}) \) in that case. The reader can easily verify this to be true.

4.1.13 DEFINITION Let \( \beta \) be a variable assignment relative to a \( \Sigma \)-structure \( \mathfrak{A} \), and let \( x_i^{u_i} \in \text{VAR}_{u_i} \) and \( d_i \in (\mathfrak{A}_u)_\infty \) for \( 1 \leq i \leq m \). Furthermore, let \( x_i^{u_i} \neq x_j^{u_j} \) whenever \( 1 \leq i < j \leq m \). Then \( [x_1^{u_1} \mapsto d_1, \ldots, x_m^{u_m} \mapsto d_m] \) is called an assignment update. Its effect \( \beta[x_1^{u_1} \mapsto d_1, \ldots, x_m^{u_m} \mapsto d_m] \) on \( \beta \) is defined by:

\[
\text{DOM}(\beta[x_1^{u_1} \mapsto d_1, \ldots, x_m^{u_m} \mapsto d_m]^v) = \text{DOM}(\beta^v) \setminus \{ x_i^{u_i} \mid u_i = v \land d_i = \infty \land 1 \leq i \leq m \}
\cup \{ x_i^{u_i} \mid u_i = v \land d_i \neq \infty \land 1 \leq i \leq m \},
\]

and

\[
\beta[x_1^{u_1} \mapsto d_1, \ldots, x_m^{u_m} \mapsto d_m](y^v) = \begin{cases} \ d_1 & \text{if } y^v = x_1^{u_1}, \\ \vdots & \vdots \\ d_m & \text{if } y^v = x_m^{u_m}, \\ \beta(y^v) & \text{otherwise}, \end{cases}
\]

for all \( v \in \text{SORT} \) and \( y^v \in \text{VAR}_v \).

4.1.14 DEFINITION Let a signature \( \Sigma \) and an \( \Sigma \)-interpretation \( (\mathfrak{A}, \beta) \) be given. The semantic value \( \varphi^{\alpha, \beta} \) of a formula \( \varphi \in \text{FORM} \) is such that \( \varphi^{\alpha, \beta} \in \{ 0, 1 \} \) and is recursively defined by the equations below.

\[
\text{tt}^{\alpha, \beta} = 1,
\]

\[
\text{ff}^{\alpha, \beta} = 0,
\]

\[
(\downarrow t)^{\alpha, \beta} = \begin{cases} 1 & \text{if } t^{\alpha, \beta} \neq \infty, \\ 0 & \text{otherwise}, \end{cases}
\]

\[
(s = t)^{\alpha, \beta} = \begin{cases} 1 & \text{if } s^{\alpha, \beta} = t^{\alpha, \beta} \text{ and } s^{\alpha, \beta} \neq \infty, \\ 0 & \text{otherwise}, \end{cases}
\]

\[
(P t)^{\alpha, \beta} = \begin{cases} 1 & \text{if } t^{\alpha, \beta} \in P^{\alpha}, \\ 0 & \text{otherwise}, \end{cases}
\]

\[
(-\psi)^{\alpha, \beta} = 1 - \psi^{\alpha, \beta},
\]
\[
\begin{align*}
(\psi \land \chi)^{\mathfrak{A},\beta} &= \min (\psi^{\mathfrak{A},\beta}, \chi^{\mathfrak{A},\beta}), \\
(\psi \lor \chi)^{\mathfrak{A},\beta} &= \max (\psi^{\mathfrak{A},\beta}, \chi^{\mathfrak{A},\beta}), \\
(\psi \rightarrow \chi)^{\mathfrak{A},\beta} &= \max (1 - \psi^{\mathfrak{A},\beta}, \chi^{\mathfrak{A},\beta}), \\
(\psi \leftrightarrow \chi)^{\mathfrak{A},\beta} &= \begin{cases} 
1 & \psi^{\mathfrak{A},\beta} = \chi^{\mathfrak{A},\beta}, \\
0 & \text{otherwise},
\end{cases} \\
(\forall x^u \psi)^{\mathfrak{A},\beta} &= \begin{cases} 
1 & \psi^{\mathfrak{A},\beta}[x^u \mapsto d] = 1 \text{ for all } d \in \mathfrak{A}_u, \\
0 & \text{otherwise},
\end{cases} \\
(\exists x^u \psi)^{\mathfrak{A},\beta} &= \begin{cases} 
1 & \psi^{\mathfrak{A},\beta}[x^u \mapsto d] = 1 \text{ for some } d \in \mathfrak{A}_u, \\
0 & \text{otherwise}.
\end{cases}
\end{align*}
\]

The machinery we just introduced, enables us to define the notions of satisfaction, truth, validity or logical truth, and logical entailment.

**4.1.15 Definition** Let \( \Gamma \subseteq \text{FORM} \) and \( \varphi \in \text{FORM} \) w.r.t. a language with signature \( \Sigma \).

i. \( \varphi \) is **satisfied** by a \( \Sigma \)-interpretation \( \langle \mathfrak{A}, \beta \rangle \), if and only if \( \varphi^{\mathfrak{A},\beta} = 1 \). Notation: \( \langle \mathfrak{A}, \beta \rangle \models \varphi \).

ii. \( \Gamma \) is **satisfied** by a \( \Sigma \)-interpretation \( \langle \mathfrak{A}, \beta \rangle \), if and only if \( \langle \mathfrak{A}, \beta \rangle \models \psi \) for all \( \psi \in \Gamma \). Notation: \( \langle \mathfrak{A}, \beta \rangle \models \Gamma \).

iii. \( \varphi \) is **true** in a \( \Sigma \)-structure \( \mathfrak{A} \), if and only if \( \langle \mathfrak{A}, \beta \rangle \models \varphi \) for all variable assignments \( \beta \). Notation: \( \mathfrak{A} \models \varphi \).

iv. \( \varphi \) is **valid** or logically **true**, if and only if \( \mathfrak{A} \models \varphi \) for all \( \Sigma \)-structures \( \mathfrak{A} \). Notation: \( \models \varphi \).

v. \( \Gamma \) **logically entails** \( \varphi \), or, equivalently, \( \varphi \) is a logical **consequence** of \( \Gamma \), if and only if \( \langle \mathfrak{A}, \beta \rangle \models \Gamma \) implies \( \langle \mathfrak{A}, \beta \rangle \models \varphi \) for all \( \Sigma \)-interpretations \( \langle \mathfrak{A}, \beta \rangle \). Notation: \( \Gamma \models \varphi \).

Just like in ordinary predicate logic we have that the denotation of an (extended) term or formula only depends on the denotations of the individual constants, function and predicate symbols, and the free variables occurring in that term or formula. This property is expressed in the following theorem.

**4.1.16 Convention** We will use \( I, J, \ldots \) to range over \( \Sigma \)-interpretations.

**4.1.17 Theorem** Let \( I = \langle \mathfrak{A}, \beta \rangle \) and \( J = \langle \mathfrak{B}, \gamma \rangle \) be \( \Sigma \)-interpretations such that \( \mathfrak{A}_u = \mathfrak{B}_u \) for any \( u \in \text{SORT} \). Furthermore, suppose that \( s^I = s^J \) and \( (x^u)^I = (x^u)^J \) for any symbol \( s \) and free variable \( x^u \) occurring in \( t \) and \( \varphi \). Then:

\[
I \models t^I = t^J, \\
I \models \varphi \iff J \models \varphi.
\]

**Proof** By structural induction. □
4.2  Operations on interpretations

In order to describe the effect of general substitution, and, also, the operational semantics of evolving algebras (see Chapter 5), we need some basic operations on structures. These operations are modify, extend and remove.

Modifying a structure consists of changing the interpretation of a function symbol. The effect of extending a structure is that some designated universe is extended by a new element. Removal is the opposite operation of extension: now a designated element is deleted from the appropriate universe. In this case also some necessary updates are made which render the denotation of terms referring to the designated element undefined and which restrict the domains of all functions to the new situation.

Later on we will see how the effect of an evolving algebra can be expressed with the help of these basic operations. We will now consider the operations on structures in more detail.

A modification operator is the semantic counterpart of a (general) substitution. Just like substitutions are sets of simple substitutions, modification operators are sets of local modification operators. The effect of a local modification operator on a structure is, as the name suggests, a local modification of a function or constant on the structure. Here, ‘local’ means that the value of a function at some argument is changed, or that the value of a constant is changed. A modification operator changes a structure at several points at a time.

4.2.1 Definition Let $\mathfrak{A}$ be a $\Sigma$-structure, and let for some $m > 0$ a set of $m$ function symbols or individual constants $f_i \in \text{SYM}$ with arity $\sigma(f_i) = (w_i, u_i)$ be given. Furthermore, let $d_i \in \mathfrak{A}^{w_i}$ and $e_i \in (\mathfrak{A}_{u_i})_{\infty}$ for $1 \leq i \leq m$, and suppose that the following condition holds:

$$\forall i, j \cdot [1 \leq i < j \leq m \quad \& \quad f_i \equiv f_j \implies d_i \neq d_j].$$

Then $M = [f_1[d_1 \mapsto e_1], \ldots, f_m[d_m \mapsto e_m]]$ is called a modification operator. Moreover, it is defined to be the set of local modification operators $[f_i[d_i \mapsto e_i]]$, also written as $f_i[d_i \mapsto e_i]$, where $1 \leq i \leq m$. Note that local modification operators are special modification operators, viz. for which $m = 1$.

The effect of the modification operator $M$ on $\mathfrak{A}$ is the $\Sigma$-structure $\mathfrak{A}_M$ such that:

$$\mathfrak{A}_M = \mathfrak{A}, \quad (u \in \text{SORT}),$$

$$g^{\mathfrak{A}_M} = g^x, \quad (g \in \text{SYM} \setminus \{f_1, \ldots, f_m\}),$$

and for all $f \in \{f_1, \ldots, f_m\}$:

$$\text{DOM}(f^{\mathfrak{A}_M}) = \text{DOM}(f^x) \setminus \{d \mid f[d \mapsto \infty] \in M\}$$

$$\cup \{d \mid f[d \mapsto e] \in M \text{ for some } e \neq \infty\},$$

$$f^{\mathfrak{A}_M}x = \begin{cases} e & \text{if } x = d \text{ and } f[d \mapsto e] \in M \text{ for some } e, \\ f^x & \text{otherwise.} \end{cases}$$
4.2.2 CONVENTION If $c \in \text{CON}_u$ and $e \in (\mathfrak{A}_u)_\infty$, we will write $[c \mapsto e]$ instead of $c[0 \mapsto e]$.

It follows from the definition above that modification operators can be seen as parallel compositions of local modification operators. The condition which is satisfied for modification operators is such that the local modifications are independent from each other. They could be carried out simultaneously or in any different order.

4.2.3 PROPOSITION Let a signature $\Sigma$ and a $\Sigma$-structure $\mathfrak{A}$ be given. Furthermore, let $[f_1[\mathfrak{d}_1 \mapsto e_1], \ldots, f_m[\mathfrak{d}_m \mapsto e_m]]$ be a modification operator on $\mathfrak{A}$. Then:

$$\mathfrak{A}[f_1[\mathfrak{d}_1 \mapsto e_1], \ldots, f_m[\mathfrak{d}_m \mapsto e_m]]$$
$$= \mathfrak{A}[f_{i_1}[\mathfrak{d}_{i_1} \mapsto e_{i_1}], \ldots, f_{i_m}[\mathfrak{d}_{i_m} \mapsto e_{i_m}]]$$

where $i_1 \ldots i_m$ is an arbitrary permutation of $1 \ldots m$.

PROOF The validity of this proposition is an immediate consequence of the condition imposed on modification operators (see Definition 4.2.1 above). ⊡

The next operator we will consider is the extend operator. The effect of this one is simple: it extends some designated universe with some new element.

4.2.4 DEFINITION Let a $\Sigma$-structure $\mathfrak{A}$ be given. Furthermore, let $u \in \text{SORT}$ and $d$ be some object such that $d \notin \mathfrak{A}_u$. Then $\mathfrak{E}^u_d$ is called an extend operator. Its effect on $\mathfrak{A}$ is the $\Sigma$-structure $\mathfrak{A}$$^\mathfrak{E}_d^u$ for which:

$$(\mathfrak{A}$$^\mathfrak{E}_d^u)_u = \mathfrak{A}_u \cup \{d\},$$

$$(\mathfrak{A}$$^\mathfrak{E}_d^u)_v = \mathfrak{A}_v \quad (v \in \text{SORT} \setminus \{u\}),$$

$$f^\mathfrak{E}_d^u = f^\mathfrak{A} \quad (f \in \text{SYM}).$$

4.2.5 REMARK If it is necessary to fully formalize the extend operator, a countably infinite set of reserve elements is needed from which the new elements $d$ are taken for each extend operation on an structure $\mathfrak{A}$. The set of these reserve elements has to be disjoint from the universes of $\mathfrak{A}$. We will implicitly assume some set of reserve elements with this property.

Moreover, the exact nature of the reserve elements does not matter. They are just ‘points’ which are added in the process of extending an structure. In fact, we have the following proposition, the proof of which is trivial.

4.2.6 PROPOSITION If $\mathfrak{A}$ is a $\Sigma$-structure, and $\mathfrak{E}^u_d$ and $\mathfrak{E}^u_c$ are both extend operators on $\mathfrak{A}$, then $\mathfrak{A}$$^\mathfrak{E}_d^u$ and $\mathfrak{A}$$^\mathfrak{E}_c^u$ are isomorphic.

The third and last operator of our list is the remove operator. Although its effect is not hard to understand—the effect being removing a designated element from a designated universe—, its formal description looks complicated.
4.2. Operations on interpretations

This is caused by the fact that we have to account for the functions, constants and predicates which have the element to be removed in their domain, or as their function value. As a consequence the domains of these functions, constants and predicates have to be adjusted to the new situation in which the element to be removed is not there anymore.

4.2.7 Definition Let a $\Sigma$-structure $\mathfrak{A}$ be given. Furthermore, let $u \in \text{SORT}$ and $d \in \mathfrak{A}_u$. Then $D^u_d$ is called a remove operator. Its effect on $\mathfrak{A}$ is the $\Sigma$-structure $\mathfrak{A}D^u_d$ for which:

$$(\mathfrak{A}D^u_d)_u = \mathfrak{A}_u \setminus \{d\},$$

$$(\mathfrak{A}D^u_d)_v = \mathfrak{A}_v \quad (v \in \text{SORT} \setminus \{u\}),$$

such that for all $f \in \text{SYM}$ with $\sigma(f) = (w, u)$:

$$\text{Dom}(f^{\mathfrak{A}D^u_d}) = (\text{Dom}(f^\mathfrak{A}) \cap (\mathfrak{A}D^u_d)\mathfrak{A}) \setminus \{e \in \mathfrak{A} \mid f^\mathfrak{A}e = d\},$$

$$f^{\mathfrak{A}D^u_d}x = f^\mathfrak{A}x,$$

and such that for all $g \in \text{SYM}$ with $\sigma(g) = (w, v)$ for some $v \in \text{SORT} \setminus \{u\}$:

$$\text{Dom}(g^{\mathfrak{A}D^u_d}) = \text{Dom}(g^\mathfrak{A}) \cap (\mathfrak{A}D^u_d)\mathfrak{A},$$

$$g^{\mathfrak{A}D^u_d}x = g^\mathfrak{A}x,$$

and, finally, such that for all $P \in \text{PRE}_w$:

$$P^{\mathfrak{A}D^u_d} = P^\mathfrak{A} \cap (\mathfrak{A}D^u_d)\mathfrak{A}.$$

The following two propositions state some elementary properties of the extend and remove operators. Their proofs are immediate.

4.2.8 Proposition Let a $\Sigma$-structure $\mathfrak{A}$ be given. Furthermore, let $u, v \in \text{SORT}$, $d \notin \mathfrak{A}_u$, and $e \notin \mathfrak{A}_v$, then:

$$\mathfrak{A}E^u_dE^v_e = \mathfrak{A}E^v_eE^u_d \quad \text{if} \ u \neq v \ \text{or} \ d \neq e,$$

$$\mathfrak{A}E^u_dD^u_d = \mathfrak{A}.$$  

4.2.9 Proposition Let a $\Sigma$-structure $\mathfrak{A}$ be given. Furthermore, let $u, v \in \text{SORT}$, $d \in \mathfrak{A}_u$, and $e \in \mathfrak{A}_v$, then:

$$\mathfrak{A}D^u_dD^v_e = \mathfrak{A}D^v_eD^u_d \quad \text{if} \ u \neq v \ \text{or} \ d \neq e.$$  

Removing an element from a universe has also effect on variable assignments. This effect is described by the restriction operator, which is the subject of the next definition.

4.2.10 Definition Let a $\Sigma$-interpretation $\langle \mathfrak{A}, \beta \rangle$ be given. Furthermore, let $u \in \text{SORT}$ and $d \in \mathfrak{A}_u$. Then $\triangleright^u_d$ is called a restriction operator. Its effect on $\beta$ is the variable assignment $\beta \triangleright^u_d$ for which:

$$\text{Dom}((\beta \triangleright^u_d)^u) = \text{Dom}(\beta^u) \setminus \{x^u \mid \beta(x^u) = d\},$$

$$\text{Dom}((\beta \triangleright^u_d)^v) = \text{Dom}(\beta^v),$$
for all \( v \in \text{SORT} \) such that \( v \neq u \), and

\[
\beta \uparrow^u_d (y^v) = \beta(y^v),
\]

for all \( v \in \text{SORT} \) and \( y^v \in \text{VAR}_v \).

4.2.11 Definition Let \( <A, \beta> \) be a \( \Sigma \)-interpretation. Furthermore, let \( M_1 \) be a modification operator, \( E^u_d \) be an extend operator, \( D^u_d \) be a remove operator, \( M_2 \) be an assignment update, and let all of them be defined with respect to \( <A, \beta> \). Then their effect on \( <A, \beta> \) is defined by:

\[
\begin{align*}
<A, \beta>M_1 &= <AM_1, \beta>, \\
<A, \beta>E^u_d &= <AE^u_d, \beta>, \\
<A, \beta>D^u_d &= <AD^u_d, \beta \uparrow^u_d>, \\
<A, \beta>M_2 &= <A, \beta M_2>.
\end{align*}
\]

We will now investigate the relation between the operators on interpretations we just introduced and the denotations of extended terms or formulae. One of these results is known as the substitution theorem. Our result is more general since it holds for general substitutions, provided that some consistency condition for this substitution holds.

Intuitively, a general substitution is consistent with respect to some interpretation \( I \) if the simple substitutions of that substitution do not interfere. In particular, if \( [f; s_1 := t_1] \) and \( [f; s_2 := t_2] \) are both elements of a substitution, then the following must hold:

\( I \models s_1 = s_2 \rightarrow t_1 \approx t_2 \).

But this is not sufficient, we also want the arguments at which \( f \) is to be updated, to be defined. So, it also must be the case that:

\( I \models \downarrow s_1 \land \downarrow s_2 \).

This notion of consistency is exactly what we referred to in Remark 3.2.3, where we stated the possibility of a substitution being inconsistent.

4.2.12 Definition Let \( I \) be a \( \Sigma \)-interpretation and let \( \sigma \) be a substitution of the following form:

\[
\sigma = [x_1^{u_1} := r_1, \ldots, x_k^{u_k} := r_k, c_1 := s_1, \ldots, c_m := s_m, \\
f_1 s_1 := t_1, \ldots, f_n s_n := t_n].
\]

Then \( \sigma \) is called consistent with respect to \( I \) if the following condition holds:

\( I \models \bigwedge_{i=1}^n \downarrow s_i \land \bigwedge_{f_i \neq f_j} (s_i = s_j \rightarrow t_i \approx t_j) \).
4.2. Operations on interpretations

How substitutions which are consistent with respect to some interpretation \( \mathcal{I} \) relate to modifications of \( \mathcal{I} \) is subject of the following definition and proposition.

4.2.13 Convention Let \( \sigma \) be a substitution over a signature \( \Sigma \), then \( \sigma_V \) will be used to denote the set of those simple substitutions in \( \sigma \) which have a variable as left hand side, and \( \sigma_\Sigma \) will be used to denote \( \sigma \setminus \sigma_V \).

4.2.14 Definition Let a \( \Sigma \)-interpretation \( \mathcal{I} \) be given. Furthermore, let \( \sigma \) be a substitution such that \( \sigma \) is consistent with respect to \( \mathcal{I} \), and:

\[
\sigma_V = [x_1^{u_1} := r_1, \ldots, x_k^{u_k} := r_k],
\]
\[
\sigma_\Sigma = [c_1 := s_1, \ldots, c_m := s_m, f_1 s_1 := t_1, \ldots, f_n s_n := t_n].
\]

Then the semantic counterparts of \( \sigma_\Sigma \) and \( \sigma_V \) with respect to \( \mathcal{I} \) are the modification operator \( \sigma^T_\Sigma \), and the assignment update \( \sigma^T_V \), respectively, which are defined by:

\[
\sigma^T_\Sigma = [x_1^{u_1} \mapsto r_1^T, \ldots, x_k^{u_k} \mapsto r_k^T],
\]
\[
\sigma^T_V = [c_1 \mapsto s_1^T, \ldots, c_m \mapsto s_m^T, f_1 s_1^T \mapsto t_1^T, \ldots, f_n s_n^T \mapsto t_n^T].
\]

Of course it has to be checked whether the definition above is correct. The next proposition the proof of which is a direct consequence of Definitions 3.2.2, 4.1.13 and 4.2.1, says this indeed to be the case.

4.2.15 Proposition Let a \( \Sigma \)-interpretation \( \mathcal{I} \) be given and let \( \sigma \) be a substitution such that \( \sigma \) is consistent with respect to \( \mathcal{I} \). Then \( \sigma^T_\Sigma \) is a modification operator and \( \sigma^T_V \) is an assignment update, both with respect to \( \mathcal{I} \).

4.2.16 Definition Let \( \mathcal{I} \) be a \( \Sigma \)-interpretation, and let \( \sigma \) be a substitution such that \( \sigma \) is consistent with respect to \( \mathcal{I} \), then \( \sigma^T \) is defined by:

\[
\mathcal{I}\sigma^T = \mathcal{I}\sigma^T_\Sigma\sigma^T_V.
\]

If \( \mathcal{I} = \langle \mathfrak{A}, \beta \rangle \) is a \( \Sigma \)-interpretation then the definition above implies that \( \mathcal{I}\sigma^T = \langle \mathfrak{A}\sigma^T_\Sigma, \beta\sigma^T_V \rangle \). This follows from Definition 4.2.11.

4.2.17 Theorem Substitution

Let a \( \Sigma \)-interpretation \( \mathcal{I} \) be given. Furthermore, let \( \sigma \) be a substitution such that \( \sigma \) is consistent with respect to \( \mathcal{I} \). Then for all extended function symbols \( f \in \text{Xfun}_{w,u} \) and all terms \( t \in \text{Xterm}_u \) (\( w \in \text{SORT}^+ \) and \( u \in \text{SORT} \)), and all formulae \( \varphi \) from \( \text{FORM} \):

\[
(f\sigma)^\mathcal{I} = f^{\mathcal{I}\sigma^\mathcal{I}},
\]
\[
(t\sigma)^\mathcal{I} = t^{\mathcal{I}\sigma^\mathcal{I}},
\]

\( \mathcal{I} \models \varphi\sigma \iff \mathcal{I}\sigma^T \models \varphi \).

Proof We will start with the first two parts of the theorem, which we will prove by simultaneous induction. Let \( \mathcal{I} = \langle \mathfrak{A}, \beta \rangle \). Then we have to consider the following possibilities:
• \( t \equiv \bot \). The theorem now follows from the fact that \( \bot \sigma \equiv \bot \) and \( \bot^J = \infty \) for all \( \Sigma \)-interpretations \( J \).

• \( t \equiv x^v \). Now, either \( x^v \) is affected by \( \sigma \), or not.

  - If \( x^v \) is affected by \( \sigma \), then \([x^v := s] \in \sigma \) for some term \( s \). This implies that \( x^v \sigma \equiv s \). Moreover, \( (x^v \sigma)^I = s^I = \beta \sigma^I_T(x^v) = (x^v)^I \sigma^I \).

  - If \( x^v \) is not affected by \( \sigma \), then it cannot be the case that \([x^v := s] \in \sigma \) for some term \( s \). This means that \( x^v \sigma \equiv x^v \) and \( \beta \sigma^I_T(x^v) = \beta(x^v) \), from which the result follows.

• \( t \equiv c \). Similar argument as in the preceding case.

• \( f \in \text{FUN}_{w,u} \). There are two possibilities:

  - \( f \sigma \equiv f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \). Then by Definition 4.1.11:

    \[
    f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^I \sigma(x) = \begin{cases} 
    s_1^I & \text{if } x = r_1^I, \\
    \vdots & \vdots \\
    s_m^I & \text{if } x = r_m^I, \\
    f^\alpha & \text{otherwise}
    \end{cases}
    \]

    for all \( x \in \text{DOM}(f \sigma^I) \). On the other hand, by Definition 4.2.1 we have:

    \[
    f^I \sigma^I(x) = \begin{cases} 
    s_1^I & \text{if } x = r_1^I, \\
    \vdots & \vdots \\
    s_m^I & \text{if } x = r_m^I, \\
    f^\alpha & \text{otherwise}
    \end{cases}
    \]

    for all \( x \in \text{DOM}(f^I \sigma^I) \). The right hand sides of the equations above clearly coincide. But we still have to check that the domains of \( f \sigma^I \) and \( f^I \sigma^I \) are equal. The reader has only to check the relevant definitions to see that this is indeed the case. Here the fact has to be used that \( \sigma \) is consistent with respect to \( I \). This consistency also implies that \( \sigma^I_T \) is a modification operator, which is needed in deriving the second equation above.

  - \( f \sigma \equiv f \). Easy.

• \( f \equiv g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \in X\text{FUN}_{w,u} \). We infer:

    \[
    g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m] \sigma \equiv g\sigma[r_1 \sigma \mapsto s_1 \sigma, \ldots, r_m \sigma \mapsto s_m \sigma].
    \]
By Definition 4.1.11 we have:

\[ g_\sigma[r_1 \sigma \mapsto s_1, \ldots, r_m \sigma \mapsto s_m]^{x} \sigma^I = \begin{cases} 
    s_1^{\sigma^I} & \text{if } x = r_1^{\sigma^I}, \\
    \vdots & \vdots \\
    s_m^{\sigma^I} & \text{if } x = r_m^{\sigma^I}, \\
    g^{\sigma^I} x & \text{otherwise.}
\end{cases} \]

The induction hypotheses give us:

\[ g_\sigma[r_1 \sigma \mapsto s_1, \ldots, r_m \sigma \mapsto s_m]^{x} \sigma^I = \begin{cases} 
    s_1^{\sigma^I} & \text{if } x = r_1^{\sigma^I}, \\
    \vdots & \vdots \\
    s_m^{\sigma^I} & \text{if } x = r_m^{\sigma^I}, \\
    g^{\sigma^I} x & \text{otherwise.}
\end{cases} \]

On the other hand:

\[ g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{x} \sigma^I = \begin{cases} 
    s_1^{\sigma^I} & \text{if } x = r_1^{\sigma^I}, \\
    \vdots & \vdots \\
    s_m^{\sigma^I} & \text{if } x = r_m^{\sigma^I}, \\
    g^{\sigma^I} x & \text{otherwise.}
\end{cases} \]

The right hand sides of the last two equations clearly coincide. It also follows from the induction hypotheses that the domains \( \text{Dom}(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{x} \sigma^I) \) and \( \text{Dom}(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{x} \sigma^I) \) are the same. The proof of this is straightforward.

- \( t = f s \). From the induction hypotheses for \( f \) and \( t \) we derive that:

\[ s^{\sigma^I} \in \text{Dom}(f^{\sigma^I}) \iff s^{\sigma^I} \in \text{Dom}(f^{\sigma^I}). \]

Now, suppose that \( s^{\sigma^I} \in \text{Dom}(f^{\sigma^I}) \), then by the induction hypotheses and the observation above we have:

\[ (f s)^{\sigma^I} = (f^{\sigma^I})(s^{\sigma^I}) = (f^{\sigma^I})(s^{\sigma^I}) = f^{\sigma^I} s^{\sigma^I} = (f s)^{\sigma^I}. \]

If on the other hand \( s^{\sigma^I} \notin \text{Dom}(f^{\sigma^I}) \), then:

\[ (f s)^{\sigma^I} = (f^{\sigma^I})(s^{\sigma^I}) = \infty = (f s)^{\sigma^I}. \]

This completes the proof of the first two parts of the theorem. As to the third and last part, we will use induction on the structure of \( \varphi \).
• \( \varphi \equiv \text{tt} \) or \( \varphi \equiv \text{ff} \). Trivial.

• \( \varphi \equiv \downarrow t \). Using the induction hypothesis we derive:

\[
I \models (\downarrow t)\sigma \quad \iff \quad I \models \downarrow (t\sigma)
\]
\[
\iff (t\sigma)^I \neq \infty
\]
\[
\iff tI_\sigma^I \neq \infty
\]
\[
\iff I_\sigma^I \models \downarrow t.
\]

• \( \varphi \equiv (s = t) \). Using the induction hypothesis we derive:

\[
I \models (s = t)\sigma \quad \iff \quad I \models (s\sigma = t\sigma)
\]
\[
\iff s\sigma^I = t\sigma^I \neq \infty
\]
\[
\iff sI_\sigma^I = tI_\sigma^I \neq \infty
\]
\[
\iff I_\sigma^I \models (s = t).
\]

• \( \varphi \equiv P_t \). Similar.

• \( \varphi \equiv \neg \psi \). Using the induction hypothesis we derive:

\[
I \models (\neg \psi)\sigma \quad \iff \quad I \models \neg (\psi\sigma)
\]
\[
\iff \sim I \models \psi\sigma
\]
\[
\iff \sim I_\sigma^I \models \psi
\]
\[
\iff I_\sigma^I \models \neg \psi.
\]

• \( \varphi \equiv (\psi \ast \chi) \). Straightforward.

• \( \varphi \equiv (\forall u\psi) \). We consider four cases (see Definition 3.2.10):

  - \( \sigma \) does not affect \( x^u \) and \( x^u \notin \text{FV}(\sigma) \).

  This implies that \( \sigma^{I[x^u \mapsto d]} = \sigma^I \) and that \( I[x^u \mapsto d]\sigma^I = I\sigma^I[x^u \mapsto d] \) for all \( d \in \mathbb{A}_u \). Using these facts and the induction hypothesis we infer:

\[
I \models (\forall u\psi)\sigma \quad \iff \quad I \models \forall u(\psi\sigma)
\]
\[
\iff \forall d \in \mathbb{A}_u \cdot I[x^u \mapsto d] \models \psi\sigma
\]
\[
\iff \forall d \in \mathbb{A}_u \cdot I[x^u \mapsto d]\sigma^{I[x^u \mapsto d]} \models \psi
\]
\[
\iff \forall d \in \mathbb{A}_u \cdot I[x^u \mapsto d]\sigma^I \models \psi
\]
\[
\iff \forall d \in \mathbb{A}_u \cdot I\sigma^I[x^u \mapsto d] \models \psi
\]
\[
\iff I\sigma^I \models \forall u\psi.
\]
4.2. Operations on interpretations

- $\sigma$ does not affect $x^u$ and $\psi$. This means that $\sigma$ does not affect $\forall x^u \psi$ either. Using this fact, Proposition 3.2.11, and Theorem 4.1.17, we infer:

$$\mathcal{I} \models (\forall x^u \psi)\sigma \iff \mathcal{I} \models \forall x^u \psi$$

$$\iff \mathcal{I}\sigma^{\mathcal{I}} \models \forall x^u \psi.$$ 

- $\sigma$ does not affect $x^u$, $x^u \in \text{Fv}(\sigma)$ and $\sigma$ affects $\psi$. Let $y^u$ be the first fresh variable. Then $\sigma^{\mathcal{I}[y^u \mapsto d]} = \sigma^{\mathcal{I}}$ and $\mathcal{I}[y^u \mapsto d]\sigma^{\mathcal{I}} = \mathcal{I}\sigma^{\mathcal{I}[y^u \mapsto d]}$ for all $d \in \mathfrak{A}_u$. Using these facts, the induction hypothesis (twicc), and Theorem 4.1.17, we infer:

$$\mathcal{I} \models (\forall x^u \psi)\sigma$$

$$\iff \mathcal{I} \models \forall y^u (\psi[x^u := y^u])\sigma$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}[y^u \mapsto d] = \psi[x^u := y^u]\sigma$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}[y^u \mapsto d]\sigma^{\mathcal{I}[y^u \mapsto d]} = \psi[x^u := y^u]$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}[y^u \mapsto d]\sigma^{\mathcal{I}} = \psi[x^u := y^u]$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}\sigma^{\mathcal{I}[y^u \mapsto d]} = \psi[x^u := y^u]$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}\mathcal{I}[y^u \mapsto d][x^u := y^u] = \psi$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}\sigma^{\mathcal{I}[y^u \mapsto d]}[x^u := y^u] = \psi$$

$$\iff \mathcal{I} \sigma^{\mathcal{I}} \models \forall x^u \psi.$$ 

- $\sigma$ does affect $x^u$, say $[x^u := t] \in \sigma$. Then $\sigma \setminus [x^u := t]$ does not affect $x^u$. Now, using the previous cases, Proposition 3.2.11, and Theorem 4.1.17, we deduce:

$$\mathcal{I} \models (\forall x^u \psi)\sigma$$

$$\iff \mathcal{I} \models (\forall x^u \psi)\sigma \setminus [x^u := t]$$

$$\iff \mathcal{I} [(\sigma \setminus [x^u := t])^{\mathcal{I}}] = \forall x^u \psi$$

$$\iff \mathcal{I} \sigma^{\mathcal{I}} \models \forall x^u \psi.$$ 

$\varphi \equiv (\exists x^u \psi)$. Analogous to the previous case.

This completes the proof of Theorem 4.2.17.

4.2.18 Corollary Let $\varphi \in \text{Form}$, $y^u \notin \text{Fv}(\varphi)$, and let $\mathcal{I}$ be a $\Sigma$-interpretation. Then:

$$\mathcal{I} \models Qx^u \varphi \iff \mathcal{I} \models Qy^u (\varphi[x^u := y^u]).$$

Proof We prove this corollary for the case that $Q \equiv \forall$. Using Theorem 4.2.17, the fact that $y^u \notin \text{Fv}(\varphi)$ and Theorem 4.1.17, we deduce:

$$\mathcal{I} \models \forall y^u (\varphi[x^u := y^u])$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}[y^u \mapsto d] = \varphi[x^u := y^u]$$

$$\iff \forall d \in \mathfrak{A}_u \cdot \mathcal{I}[y^u \mapsto d][x^u := y^u] = \varphi$$
\[\iff \forall d \in A_n. \cdot I[y^u \mapsto d][x^u \mapsto d] \models \varphi\]
\[\iff \forall d \in A_n. \cdot I[x^u \mapsto d] \models \varphi\]
\[\iff I \models \forall x^u \varphi.\]

The case that \(Q \equiv \exists\) is similar. \(\blacksquare\)

In order to state some important properties of the remove operator we need the notion of the relativization of a formula \(\varphi\) with respect to a term \(t\). The idea is that in the relativized formula the denotation of the term \(t\) is excluded from the range of the quantifiers which occur in the original formula (with respect to an interpretation).

\subsection*{4.2.19 Definition} Let \(\varphi \in \text{FORM}\) and \(t \in \text{TERM}_u\) for some \(u \in \text{SORT}\). Then the relativization \(\overline{\varphi}^t\) of \(\varphi\) with respect to \(t\) is recursively defined by:

\[
\begin{align*}
\overline{tt}^t & = tt, \\
\overline{ff}^t & = ff, \\
\overline{\overline{r}}^t & = \overline{r}, \\
\overline{(r \equiv s)}^t & = (r \equiv s), \\
\overline{Pr}^t & = Pr, \\
\overline{\neg \psi}^t & = \neg \psi^t, \\
\overline{(\psi \cdot \chi)}^t & = (\overline{\psi}^t \cdot \overline{\chi}^t), \\
\overline{\forall x^u \psi}^t & = \begin{cases} 
(\forall x^u (\neg (x^u = t) \rightarrow \overline{\psi}^t)) & \text{if } u \equiv v, \\
(\forall x^u \psi^t) & \text{if } u \neq v,
\end{cases} \\
\overline{\exists x^u \psi}^t & = \begin{cases} 
(\exists x^u (\neg (x^u = t) \land \overline{\psi}^t)) & \text{if } u \equiv v, \\
(\exists x^u \overline{\psi}^t) & \text{if } u \neq v.
\end{cases}
\end{align*}
\]

\subsection*{4.2.20 Convention} i. If \(\varphi \in \text{FORM}\) and \(w \equiv u_1 \ldots u_n \in \text{SORT}^+\), then \(\forall x^w \varphi\) denotes the formula \(\forall x^{u_1} \ldots \forall x^{u_n} \varphi\).

ii. If the types of the variables in \(x^w\) can be derived from the context, we will sometimes write \(x\) instead of \(x^w\) for reasons of simplicity of notation.

The next theorem states a property of the interpretation of terms, which can be paraphrased as follows. Let \(I = (A, \beta)\) be some \(\Sigma\)-interpretation and \(A_u\) be one of its universes. Furthermore, let \(t_I = d \in A_u\). Now, if \(s\) is a term, then \(s[t := 1]\) will have the same denotation with respect to \(I\), as \(s\) with respect to \(ID^u_I\), provided that some conditions are met. These conditions, which are slightly dependent on the form of \(t\), roughly state that \(s\) may not contain any symbols, which are different from \(t\) and do have references to \(d\). The conditions labeled 'A', 'B', and 'C' express this property in a syntactical way.
4.2.21 Theorem Let \( t \in \text{T} \text{ERM}_u \) and \( s \in \text{XTERM}_u \) for some signature \( \Sigma \) and \( u, v \in \text{SORT} \). Furthermore, let \( F \) be the set of all function symbols \( f \) occurring in \( s \) such that \( f \in \text{FUN}_{w,u} \) for some \( w \in \text{SORT}^+ \), let \( C \) be the set of all constant symbols \( c \in \text{CON}_u \) occurring in \( s \), and let \( X \) be the set of all variables \( x^u \in \text{VAR}_u \cap \text{FV}(s) \). Finally, let \( \mathcal{I} = \langle \mathfrak{A}, \beta \rangle \) be a \( \Sigma \)-interpretation. Then:

\[
s^{\text{TD}_{\mathcal{I}}}_t = s[t := \bot]^{\mathcal{I}},
\]

if one of the following conditions is satisfied:

\begin{enumerate}
\item \( t \in \text{CON}_u \cup \text{VAR}_u \), and
\[
\mathcal{I} \models \downarrow t \wedge \bigwedge_{f \in F} \forall x (fx \neq t)^t \wedge \bigwedge_{c \in \text{CON} \cup X} c \neq t,
\]

\item \( t \equiv gt \) for some \( g \notin F \), and
\[
\mathcal{I} \models \downarrow t \wedge \bigwedge_{f \in F} \forall x (fx \neq t)^t \wedge \bigwedge_{c \in \text{CONTX}} c \neq t,
\]

\item \( t \equiv gt \) for some \( g \in F \), and
\[
\mathcal{I} \models \downarrow t \wedge \bigwedge_{f \in F, f \neq g} \forall x (fx \neq t)^t \wedge \forall x (x \neq t \rightarrow gx \neq t)^t \wedge \bigwedge_{c \in \text{CONTX}} c \neq t,
\]
\end{enumerate}

provided that the variables \( x \) in the conditions above are are fresh, i.e. do not occur in \( t \) or \( t \).

Proof The proof is by simultaneous induction on the structure of extended terms \( s \) and extended function symbols \( f \in \text{XFUN}_{w,u} \). This can be done, for the theorem can be adapted to also cover extended function symbols. If one of the conditions A, B, or C holds (with respect to all function symbols, constant symbols, and variables occurring in \( f \)), then:

\[
\text{DOM}(f^{\text{TD}_{\mathcal{I}}}) = \text{DOM}(f[t := \bot]^{\mathcal{I}}) \cap (\text{AD}_{\mathcal{I}})^w,
\]

\[
f^{\text{TD}_{\mathcal{I}}}_t x = f[t := \bot]^{\mathcal{I}} x.
\]

Firstly, observe that from \( \mathcal{I} \models \downarrow t \) it follows that the operator \( D_{\mathcal{I}}^s \) is defined on \( \mathcal{I} \). Now, consider the different cases for \( s \) and \( f \), assuming that one of the appropriate conditions of the theorem is fulfilled for \( t \).

- \( s \equiv \bot \). Here, for any \( t \), we have \( s[t := \bot] \equiv s \equiv \bot \), from which the result directly follows from Definition 4.1.11.

- \( s \equiv x^v \). There are two possibilities:

  - \( t \equiv x^v \). In this case \( v \equiv u \) and \( s[t := \bot] \equiv \bot \), from which we infer \( s^{\text{TD}_{\mathcal{I}}}_t = \beta \triangleright (x^v)\mathcal{I} (x^v) = \infty = s[t := \bot]^{\mathcal{I}} \), by applying Definition 4.2.10.
\[ t \not\equiv x^v. \] Now \( x^v \in X \) or \( x^v \not\in X \). In both cases we have \( s[t:=\bot] = s \) if \( x^v \in X \), then \( v \equiv u \). Together with one of the conditions A, B or C, this implies \( I \models x^v \neq t \). Application of Definition 4.2.10 gives the desired result.

If \( x^v \not\in X \), then \( v \not\equiv u \). Again, the result follows from Definition 4.2.10.

- \( s \equiv c \). This case is very similar to the preceding one.

- \( t \equiv c \). As in the corresponding case above, we have \( s[t:=\bot] = c \), from which we derive \( s^{TD^{I^v}_c} = \infty = s[t:=\bot]^I \) using Definition 4.2.7.

- \( t \not\equiv c \). Here, \( c \in C \) or \( c \not\in C \). In both cases we infer in much the same way as we have done in the corresponding case above, that \( s^{TD^{I^v}_c} = s[t:=\bot]^I \).

- \( f \in \text{FUN}_{w,v} \). Again, there are two cases to consider:

  - \( t \equiv ft \) for some \( t \in \text{TERM}^w \). This means that \( v \equiv u \) and \( f[t:=\bot] \equiv f[t\rightarrow \bot] \). By Definition 4.2.7 we have \( \text{DOM}(f^{TD^{I^v}_c}) = (\text{DOM}(f^I) \cap (\mathcal{A}^{D^{I^v}_c})^w) \setminus \{t^I\} \) and \( f^{TD^{I^v}_c} x = f^I x \), since \( I \models \forall x(x \neq t \rightarrow f x \neq t) \) by condition C. On the other hand, by Definition 4.1.11 we have \( \text{DOM}(f[t\rightarrow \bot]^I) = \text{DOM}(f^I) \setminus \{t^I\} \), and \( f[t\rightarrow \bot]^I x = f^I x \). These observations imply that \( \text{DOM}(f^{TD^{I^v}_c}) = \text{DOM}(f[t\rightarrow \bot]^I) \cap (\mathcal{A}^{D^{I^v}_c})^w \) and \( f^{TD^{I^v}_c} x = f[t:=\bot]^I x \).

  - \( t \not\equiv ft \). Here, \( f[t:=\bot] \not\equiv f \). The result follows by applying Definition 4.2.7 and, if \( v \equiv u \), condition A or B.

- \( f \equiv g[r_1\mapsto s_1, \ldots, r_m\mapsto s_m] \) for some \( g \in \text{XFUN}_{w,v} \), some \( r_1, \ldots, r_m \in \text{XTERM}^w \), and some \( s_1, \ldots, s_m \in \text{XTERM}_v \). In the following, we will use \( g[r_1\mapsto s_1] \) as an abbreviation for \( g[r_1\mapsto s_1, \ldots, r_m\mapsto s_m] \). Now, \( f[t:=\bot] \equiv g[t:=\bot][r_1[t:=\bot] \mapsto s_1[t:=\bot]] \). Since all symbols occurring in \( g, r_1, \) and \( s_1 \) also occur in \( f \), the appropriate conditions of the theorem also hold for these. This means that \( \text{DOM}(g^{TD^{I^v}_c}) = \text{DOM}(g[t:=\bot]^I) \cap (\mathcal{A}^{D^{I^v}_c})^w \) and \( g^{TD^{I^v}_c} x = g[t:=\bot]^I x \), and, furthermore, that \( r_1^{TD^{I^v}_c} = r_1[t:=\bot] \), and \( s_1^{TD^{I^v}_c} = s_1[t:=\bot]^I \). These equalities imply that, if the condition \( C(i) \) of Definition 4.1.11 holds with respect to \( r_i[t:=\bot]^I \) and \( s_i[t:=\bot]^I \) for \( 1 \leq i \leq m \), it will also hold with respect to \( r_i^{TD^{I^v}_c} \) and \( s_i^{TD^{I^v}_c} \) (\( 1 \leq i \leq m \)). Taking also into account that \( \{r_i^{TD^{I^v}_c} \mid C(i) \land 1 \leq i \leq m\} \subseteq (\mathcal{A}^{D^{I^v}_c})^w \), and using Definition 4.1.11 again, we conclude that \( \text{DOM}(f^{TD^{I^v}_c}) = \text{DOM}(f[t:=\bot]^I) \cap (\mathcal{A}^{D^{I^v}_c})^w \) and \( f^{TD^{I^v}_c} x = f[t:=\bot]^I x \).

- \( s \equiv gr \) for some \( g \in \text{XFUN}_{w,v} \) and \( r \in \text{XTERM}^v \). In this case we have \( s[t:=\bot] = g[t:=\bot]r[t:=\bot] \). By the induction hypothesis we have \( \text{DOM}(g^{TD^{I^v}_c}) = \text{DOM}(g[t:=\bot]^I) \cap (\mathcal{A}^{D^{I^v}_c})^w \) and \( g^{TD^{I^v}_c} x = g[t:=\bot]^I x \), and also \( r^{TD^{I^v}_c} = r[t:=\bot]^I \).
From the last equation we derive that $r_i[t := \bot]^I \neq t^I$ for any component $r_i$ of $r$ such that $r_i \in \text{XTERM}_u$. Since $\text{DOM}(g^{\text{ID}^n_{i_I}}) = \text{DOM}(g[t := \bot]^I) \cap (\text{AD}^n_{v_i})^w$, this implies that $r^{\text{ID}^n_{i_I}} \in \text{DOM}(g^{\text{ID}^n_{i_I}})$ if and only if $r[t := \bot]^I \in \text{DOM}(g[t := \bot]^I)$.

Now, if $r^{\text{ID}^n_{i_I}} \in \text{DOM}(g^{\text{ID}^n_{i_I}})$, then by our observation above $r[t := \bot]^I \in \text{DOM}(g[t := \bot]^I)$. So, by using Definition 4.1.11, we may infer that $s^{\text{ID}^n_{i_I}} = g^{\text{ID}^n_{i_I}} r^{\text{ID}^n_{i_I}} = g[t := \bot]^I r[t := \bot]^I = s[t := \bot]^I$.

If, on the other hand, $r^{\text{ID}^n_{i_I}} \notin \text{DOM}(f^{\text{ID}^n_{i_I}})$, we also have $r[t := \bot]^I \notin \text{DOM}(g[t := \bot]^I)$. Again using Definition 4.1.11, this yields $s^{\text{ID}^n_{i_I}} = \infty = s[t := \bot]^I$.

This concludes the proof of Theorem 4.2.21.

Note that in the theorem we just proved, $t$ had to be a 'normal' term, i.e. $t \in \text{TERM}_u$. The reason for this is that the substitution $[t := \bot]$ occurring in the left part of the implication would not be a substitution if $t$ would be an arbitrary extended term (cf. Definition 3.2.2).

The next corollary is almost trivial. The second part of it can be seen as a variation of Theorem 4.1.17, where the interpretation of all symbols in a term remains constant, except that one universe is expanded by a new element. This corollary expresses that the interpretation does not change in such cases. It could be proved directly by induction on the structure of terms. We will use Theorem 4.2.21, however.

### 4.2.22 Corollary
Let $s \in \text{XTERM}_u$ and let $x^u \notin \text{Fv}(s)$. Furthermore, let $\mathcal{I}$ be a $\Sigma$-interpretation, and let $E_\Sigma^u$ be an extend operator with respect to $\mathcal{I}$. Moreover, let the sets $F$, $C$ and $X$ be defined for $s$ in the same way as in Theorem 4.2.21. Then the following holds:

$$TE^u_\sigma[x^u \rightarrow d] = \downarrow x^u \land \bigwedge_{f \in F} \forall x (f \neq x^u) x^u \land \bigwedge_{c \in C \cup X} c \neq x^u,$$

provided that the variables $x$ are fresh, i.e. are different from $x^u$.

**Proof** The first part of the corollary is a direct consequence of Definition 4.2.4. Moreover, it is condition A of Theorem 4.2.21 with $J = E_\sigma^u[x^u \rightarrow d]$ substituted for $\mathcal{J}$, and $x^u$ substituted for $t$. Applying the theorem for this choice yields:

$$s^I = s^{TE^u_\sigma[x^u \rightarrow d]}D^u_\sigma$$

$$= s^{TE^u_\sigma[x^u \rightarrow d]} \uparrow\uparrow_{x^u}$$

$$= s[x^u := \bot][TE^u_\sigma[x^u \rightarrow d]]$$

$$= s^{TE^u_\sigma[x^u \rightarrow d]}.$$
4.2.23 Theorem Let \( t \in \text{TERM}_u \) and \( \varphi \in \text{FORM} \) for some signature \( \Sigma \) and \( u \in \text{SORT} \). Furthermore, let \( F \) be the set of all function symbols \( f \) occurring in \( \varphi \) such that \( f \in \text{FUN}_{w,u} \) for some \( w \in \text{SORT}^+ \), let \( C \) be the set of all constant symbols \( c \in \text{CON}_u \) occurring in \( \varphi \), and let \( X \) be the set of all variables \( x^u \in \text{VAR}_u \cap \text{FV}(\varphi) \). Finally, let \( \mathcal{I} = \langle \mathcal{A}, \beta \rangle \) be a \( \Sigma \)-interpretation. Then:

\[
\mathcal{I}^t_{\mathcal{I}} \models \varphi \ \iff \ \mathcal{I} \models \varphi[t := \perp]^t,
\]

if one of the following conditions is satisfied:

A. \( t \in \text{CON}_u \cup \text{VAR}_u \), and

\[
\mathcal{I} \models \downarrow t \land \bigwedge_{f \in F} \overline{\forall x(f x \neq t)^t} \land \bigwedge_{c \in C \cup X \atop c \neq t} c \neq t,
\]

B. \( t \equiv gt \) for some \( g \not\in F \), and

\[
\mathcal{I} \models \downarrow t \land \bigwedge_{f \in F} \overline{\forall x(f x \neq t)^t} \land \bigwedge_{c \in C \cup X} c \neq t,
\]

C. \( t \equiv gt \) for some \( g \in F \), and

\[
\mathcal{I} \models \downarrow t \land \bigwedge_{f \in F \atop f \neq g} \overline{\forall x(f x \neq t)^t} \land \overline{\forall x(x \neq t \rightarrow g x \neq t)^t} \land \bigwedge_{c \in C \cup X} c \neq t,
\]

provided that the variables \( x \) in the conditions above are are fresh, i.e. do not occur in \( t \) or \( t \).

Proof The proof is by induction on the complexity of \( \varphi \). As in the proof of the previous theorem, note that \( \mathcal{I} \models \downarrow t \) implies that the operator \( \text{D}^t_{\mathcal{I}} \) is defined on \( \mathcal{I} \). Consider the following cases for \( \varphi \) assuming that one of the appropriate conditions of the theorem is fulfilled for \( \varphi \):

- \( \varphi \equiv \text{tt} \) or \( \varphi \equiv \text{ff} \). These cases are trivial.

- \( \varphi \equiv \downarrow s \). In this case we have that \( (\downarrow s)[t := \perp]^t \equiv \downarrow s[t := \perp] \). Since \( \downarrow s \) and \( s \) contain exactly the same function symbols, constants and variables, we may apply the previous theorem, which yields \( s^\mathcal{I}_t = s[t := \perp]^\mathcal{I} \). The result now follows from Definition 4.1.14.

- \( \varphi \equiv (r = s) \). Here we have \( (r = s)[t := \perp]^t \equiv (r[t := \perp] = s[t := \perp]) \). Applying Theorem 4.2.21 to \( r \) and \( s \) gives the desired result.

- \( \varphi \equiv P r \) for some \( P \in \text{PRE}_w \). Now \( (P r)[t := \perp]^t \equiv P r[t := \perp] \). Applying Theorem 4.2.21 to the components of \( r \) yields \( r^{\mathcal{I}}_t = r[t := \perp]^\mathcal{I} \). From this equation we derive that \( r_i[t := \perp]^\mathcal{I} \neq r^\mathcal{I} \) for any component \( r_i \) of \( r \) such that \( r_i \in X \text{TERM}_u \). Since \( D^{\mathcal{I}}_t = P^\mathcal{I} \cap \langle \mathcal{A} \rangle^\mathcal{I} \), we may deduce that \( r^{\mathcal{I}}_t \in P^\mathcal{I} \) if and only if \( r[t := \perp]^\mathcal{I} \in P^\mathcal{I} \), which settles this case.
4.2. Operations on interpretations

• \( \varphi \equiv \neg \psi \). From the fact that \((\neg \psi)[t:=\bot]^t \equiv \neg \psi[t:=\bot]^t\) and from the induction hypothesis, which is supposed to hold for \(\psi\), the result can be derived as follows:

\[
\mathcal{ID}_{t^2}^u \models \neg \psi \iff \sim \mathcal{ID}_{t^2}^u \models \psi \\

\iff \mathcal{I} \models \psi[t:=\bot]^t \\

\iff \mathcal{I} \models \neg \psi[t:=\bot]^t \\

\iff \mathcal{I} \models (\neg \psi)[t:=\bot]^t.
\]

• \( \varphi \equiv (\psi \star \chi) \). Here the result can be obtained from the fact that \((\psi \star \chi)[t:=\bot]^t \equiv \psi[t:=\bot]^t \star \chi[t:=\bot]^t\) and a straightforward application of the induction hypothesis, just like in the case \( \varphi \equiv \neg \psi \) above.

• \( \varphi \equiv \forall x^u \psi \). Here we have two cases: either \( v \equiv u \) or \( v \not\equiv u \).

  - \( v \equiv u \). Assume that \( x^u \notin \text{FV}(t) \) (this condition can be enforced by renaming \( x^u \) into a fresh variable using Corollary 4.2.18). This has the following consequences. Let \( d \in A_u \) such that \( d \neq t^2 \), then \( t^2 = \nu[t^2 \mapsto \psi] \) by Theorem 4.1.17, and \( \mathcal{I}[x^u \mapsto d] \models x^u \neq t \). We also have that \((\forall x^u \psi)[t:=\bot]^t \equiv \forall x^u(x^u \neq t \rightarrow \psi[t:=\bot]^t)\).

In the following we want to apply the induction hypothesis to \( \psi \) with respect to \( \mathcal{I}[x^u \mapsto d] \) where \( d \neq t^2 \). This means that one of the conditions \( A, B \) or \( C \) of the theorem should hold for \( \psi \) with respect to \( \mathcal{I}[x^u \mapsto d] \) and the sets \( F, C \) and \( X \). Now, all function symbols and constants in \( \psi \) also occur in \( \forall x^u \psi \). Moreover, all free variables in \( \psi \) are also free in \( \forall x^u \psi \), with the possible exception of \( x^u \). Applying Theorem 4.1.17 and ignoring \( x^u \) for the moment, we deduce that the same condition which holds for \( \forall x^u \psi \) with respect to \( \mathcal{I} \), also holds for \( \psi \) with respect to \( \mathcal{I}[x^u \mapsto d] \). Regarding the variable \( x^u \) which possibly occurs free in \( \psi \), we observed that \( \mathcal{I}[x^u \mapsto \neg] \models x^u \neq t \) for all \( d \neq t^2 \). This completes the condition that should be fulfilled.

The derivation runs as follows:

\[
\mathcal{ID}_{t^2}^u \models \forall x^u \psi \\

\iff \forall d \in (A\mathcal{D}_{t^2}^u)_u \cdot \mathcal{ID}_{t^2}^u[x^u \mapsto d] \models \psi \\

\iff \forall d \in A_u \setminus \{t^2\} \cdot \mathcal{I}[x^u \mapsto d] \mathcal{D}_{t^2}^u \models \psi \\

\iff \forall d \in A_u \setminus \{t^2\} \cdot \mathcal{I}[x^u \mapsto d] \mathcal{D}_{t^2}[x^u \mapsto d] \models \psi \\

\iff \forall d \in A_u \setminus \{t^2\} \cdot \mathcal{I}[x^u \mapsto d] \models \psi[t:=\bot]^t \\

\iff \mathcal{I} \models \forall x^u(x^u \neq t \rightarrow \psi[t:=\bot]^t) \\

\iff \mathcal{I} \models (\forall x^u \psi)[t:=\bot]^t.
\]
Chapter 4. The semantics of $E$-logic

$- v \neq u$. Assuming again that $x^u \notin \text{Fv}(t)$, we have that $t^\mathcal{I} = t^{\mathcal{I}[x^u \mapsto d]}$, by Theorem 4.1.17. Moreover, we have in this case that $(\forall x^u \psi)[t := \bot]^t = \forall x^u \psi[t := \bot]^t$.

Now, we derive, partly using the same arguments as in the preceding case:

\[ \mathcal{I}D^u_t \models \forall x^u \psi \iff \forall d \in \mathcal{A}_v \cdot \mathcal{I}D^u_t[x^u \mapsto d] \models \psi \]
\[ \iff \forall d \in \mathcal{A}_v \cdot \mathcal{I}[x^u \mapsto d]D^u_{\mathcal{I}} \models \psi \]
\[ \iff \forall d \in \mathcal{A}_v \cdot \mathcal{I}[x^u \mapsto d]D^u_{\mathcal{I}[x^u \mapsto d]} \models \psi \]
\[ \iff \forall d \in \mathcal{A}_v \cdot \mathcal{I}[x^u \mapsto d] \models \psi[t := \bot]^t \]
\[ \iff \mathcal{I} \models \forall x^u \psi[t := \bot]^t \]
\[ \iff \mathcal{I} \models (\forall x^u \psi)[t := \bot]^t. \]

$\bullet \ \varphi \equiv \exists x^u \psi$. Analogous to the previous case where $\varphi \equiv \forall x^u \psi$.

This completes the proof of Theorem 4.2.23. $\blacksquare$

The next corollary is the analogue of Corollary 4.2.22 for formulae. In this case the result is not so obvious, however. Formulae may contain quantifiers. So, expanding a universe with a new element might have an effect on the truth value of a formula. The corollary accounts for this situation by stipulating that in such an expanded interpretation a formula has to be relativized with respect to the new element in order that the formula keeps its truth value. Of course, the interpretation of all other symbols in the formula have to remain unaltered.

4.2.24 COROLLARY Let $\varphi \in \text{FORM}$ and let $x^u \notin \text{Fv}(\varphi)$. Furthermore, let $\mathcal{I}$ be a $\Sigma$-interpretation, and let $E^u_d$ be an extend operator with respect to $\mathcal{I}$. Then the following holds:

\[ \mathcal{I} \models \varphi \iff \mathcal{I}E^u_d[x^u \mapsto d] \models \varphi[x^u]. \]

PROOF By Corollary 4.2.22, condition A of Theorem 4.2.23 is satisfied for the choice $\mathcal{J} = \mathcal{I}E^u_d[x^u \mapsto d]$ for $\mathcal{I}$, and $x^u$ for $t$. Applying the theorem now, we derive:

\[ \mathcal{I} \models \varphi \iff \mathcal{I}E^u_d[x^u \mapsto d]D^u_{\mathcal{J}} \models \varphi \]
\[ \iff \mathcal{I}E^u_d[x^u \mapsto d]D^u_{\mathcal{J}[x^u]} \models \varphi \]
\[ \iff \mathcal{I}E^u_d[x^u \mapsto d] \models \varphi[x^u := \bot]^t \]
\[ \iff \mathcal{I}E^u_d[x^u \mapsto d] \models \varphi[x^u]. \]

In the first step of this derivation we used Theorem 4.1.17, and Definitions 4.2.4, 4.2.7 and 4.2.10. In the last step, we used the fact that $x^u \notin \text{Fv}(\varphi)$. $\blacksquare$
4.3 Soundness and completeness

In this section we will prove that our $E$-logic is sound and complete. First, we will prove that the axioms are valid and that the derivation rules are truth preserving. Then soundness will be proven by induction on the length of an arbitrary derivation. Here, the truth preserving property of the derivation rules plays a crucial role.

Having proven soundness of our logic, we will concentrate on completeness. Our proof will be a rather straightforward extension of the Lindenbaum-Henkin construction. In our case, we have to cope with extended function symbols and extended terms.

4.3.1 Definition Let $R$ be a derivation rule different from $Rei$, having as its premises:

$$
\begin{array}{c}
\varphi_1, \ldots, \varphi_m, \quad \Phi_1, \ldots, \Phi_n, \\
\vdash \psi_1 \\
\vdash \psi_n
\end{array}
$$

and as conclusion $\chi$. Then $R$ is called truth preserving if for any $\Gamma \subseteq \text{FORM}$:

$$
\begin{array}{c}
\Gamma \models \varphi_1 \quad \& \quad \ldots \quad \& \quad \Gamma \models \varphi_m \\
\quad \& \quad \Gamma \cup \Phi_1 \models \psi_1 \quad \& \quad \ldots \quad \& \quad \Gamma \cup \Phi_n \models \psi_n \\
\Rightarrow \quad \Gamma \models \chi,
\end{array}
$$

provided that, in case of $\forall I$ and $\exists E$, the required side conditions on variables hold with respect to $\Gamma$, instead of the active hypotheses (see Definition 3.3.5).

The rule $Rei$ is excluded from the definition above for the following reason. It is a so-called structural rule. It describes how formulae that have already been derived in a derivation, may be used in the remainder of it. So, in contrast to the other derivation rules, $Rei$ is not a rule about a logical construction provided by a logical connective or by an extended function symbol. In fact, $Rei$ states a property of $E$-logic (and all classical logics), which is traditionally called monotonicity.

4.3.2 Proposition $E$-logic satisfies the monotonicity property, i.e. for any $\Gamma, \Delta \subseteq \text{FORM}$ and $\varphi \in \text{FORM}$, if $\Gamma \subseteq \Delta$ then:

$$
\Gamma \models \varphi \quad \Rightarrow \quad \Delta \models \varphi.
$$

Proof Monotonicity is a direct consequence of Definition 3.3.8. ■

4.3.3 Theorem All axioms of $E$-logic are valid and all its derivation rules different from $Rei$ are truth preserving.

Proof The proof for the axiom $ttA$, and the rules $ffI$, $ffE$, $\forall I$, $\forall I$, $\forall E$, $\land I$, $\land E$, $\rightarrow I$, $\rightarrow E$, $\leftrightarrow I$, $\leftrightarrow E$, and $\leftrightarrow E$ is straightforward and classical, since this axiom and these rules have nothing whatsoever to do with the partiality of the logic. We will prove some of the remaining cases.
• **∀I** In this case we have to prove:

\[ \Gamma, \downarrow y^n \models \varphi[x^n := y^n] \quad \Rightarrow \quad \Gamma \models \forall x^n \varphi, \]

under the condition that \( y^n \notin \text{Fv}(\varphi) \) or \( y^n \equiv x^n \), and \( y^n \notin \text{Fv}(\Gamma) \).

So, suppose that \( \Gamma, \downarrow y^n \models \varphi[x^n := y^n] \) and that \( (\mathfrak{A}, \beta) \models \Gamma \). Now, consider the \( \Sigma \)-interpretation \( (\mathfrak{A}, \beta[y^n \mapsto d]) \) for arbitrary \( d \in \mathfrak{A}_u \). Then, by Theorem 4.1.17 and Definition 4.1.14, \( (\mathfrak{A}, \beta[y^n \mapsto d]) \models \Gamma \cup \{ \downarrow y^n \} \).

From our assumption we deduce that also \( (\mathfrak{A}, \beta[y^n \mapsto d]) \models \varphi[x^n := y^n] \).

Since \( d \) was chosen to be arbitrary, we infer from Definition 4.1.14 \( (\mathfrak{A}, \beta) \models \forall y^n \varphi[x^n := y^n] \). Putting things together, we have shown that \( \Gamma \models \forall y^n \varphi[x^n := y^n] \), which is equivalent to \( \Gamma \models \forall x^n \varphi \), by Corollary 4.2.18, since \( y^n \notin \text{Fv}(\varphi) \) or \( y^n \equiv x^n \).

• **∀E** We have to prove that:

\[ \Gamma \models \forall x^n \varphi \quad \text{and} \quad \Gamma \models \downarrow t \quad \Rightarrow \quad \Gamma \models \varphi[x^n := t]. \]

Assume that \( \Gamma \models \forall x^n \varphi \), \( \Gamma \models \downarrow t \), and that \( (\mathfrak{A}, \beta) \models \Gamma \). This implies that \( (\mathfrak{A}, \beta) \models \forall x^n \varphi \) and \( (\mathfrak{A}, \beta) \models \downarrow t \). So, we have \( (\mathfrak{A}, \beta[x^n \mapsto d]) \models \varphi \) for all \( d \in \mathfrak{A}_u \). From \( (\mathfrak{A}, \beta) \models \downarrow t \) we deduce that \( t^{A, \beta} \in \mathfrak{A}_u \), by Definition 4.1.14.

Combining our observations yields \( (\mathfrak{A}, \beta[x^n \mapsto t^{A, \beta}]) \models \varphi \), from which we conclude \( (\mathfrak{A}, \beta) \models \varphi[x^n := t] \), by applying Theorem 4.2.17. This had to be proven.

• **∃E** We have to show that:

\[ \Gamma \models \exists x^n \varphi \quad \text{and} \quad \Gamma, \downarrow y^n, \varphi[x^n := y^n] \models \psi \quad \Rightarrow \quad \Gamma \models \psi, \]

provided that \( y^n \notin \text{Fv}(\varphi) \) or \( y^n \equiv x^n \), and \( y^n \notin \text{Fv}(\psi) \), and \( y^n \notin \text{Fv}(\Gamma) \).

Assume that \( \Gamma \models \exists x^n \varphi \), \( \Gamma, \downarrow y^n, \varphi[x^n := y^n] \models \psi \), and \( (\mathfrak{A}, \beta) \models \Gamma \). This implies \( (\mathfrak{A}, \beta) \models \exists x^n \varphi \), from which we infer \( (\mathfrak{A}, \beta) \models \exists y^n \varphi[x^n := y^n] \) by applying Corollary 4.2.18. This means that \( (\mathfrak{A}, \beta[y^n \mapsto d]) \models \varphi[x^n := y^n] \) for some \( d \in \mathfrak{A}_u \). Clearly, for this \( d \in \mathfrak{A}_u \) we have \( (\mathfrak{A}, \beta[y^n \mapsto d]) \models \downarrow y^n \).

Combining these facts with our assumption that \( \Gamma, \downarrow y^n, \varphi[x^n := y^n] \models \psi \) and the fact that \( y^n \notin \text{Fv}(\Gamma) \), gives us \( (\mathfrak{A}, \beta[y^n \mapsto d]) \models \psi \). But then, since \( y^n \notin \text{Fv}(\psi) \), we may conclude \( (\mathfrak{A}, \beta) \models \psi \), which we had to prove.

• **Sub** We have to show that:

\[ \Gamma \models s = t \quad \text{and} \quad \Gamma \models \varphi[x^n := s] \quad \Rightarrow \quad \Gamma \models \varphi[x^n := t]. \]

Assume that \( \Gamma \models s = t \), \( \Gamma \models \varphi[x^n := s] \), and \( (\mathfrak{A}, \beta) \models \Gamma \). From this we infer \( s^{A, \beta} = t^{A, \beta} \), and \( (\mathfrak{A}, \beta[x^n \mapsto s^{A, \beta}]) \models \varphi \), by Theorem 4.2.17. Combining these facts yields \( (\mathfrak{A}, \beta[x^n \mapsto t^{A, \beta}]) \models \varphi \). Again applying Theorem 4.2.17 gives us \( (\mathfrak{A}, \beta) \models \varphi[x^n := t] \), which we had to prove.
4.3. Soundness and completeness

- **fStr** We have to prove:

\[ \Gamma \models \downarrow f(t_1, \ldots, t_i, \ldots, t_n) \Rightarrow \Gamma \models \downarrow t_i. \]

Assume that \( \Gamma \models \downarrow f(t_1, \ldots, t_i, \ldots, t_n) \), and \( \langle A, \beta \rangle \models \Gamma \). This implies that \( \langle A, \beta \rangle \models \downarrow f(t_1, \ldots, t_i, \ldots, t_n) \), which, according to Definitions 4.1.14 and 4.1.11, can only be the case if \( (t_1, \ldots, t_i, \ldots, t_n)A,\beta \in \text{DOM} \ (fA,\beta) \). But this means that \( t_iA,\beta \neq \infty \), which settles this case.

- **Xf1** What we have to show, is:

\[ \Gamma \models t = r_i \quad \& \quad \Gamma \models \downarrow s_i \quad \& \quad \forall j \neq i \cdot \Gamma, t = r_j \models s_i = s_j \]

\[ \Rightarrow \Gamma \models f[r_1\mapsto s_1, \ldots, r_i\mapsto s_i, \ldots, r_m\mapsto s_m]t = s_i. \]

Suppose that \( \Gamma \models t = r_i \), \( \Gamma \models \downarrow s_i \), \( \forall j \neq i \cdot \Gamma, t = r_j \models s_i = s_j \), and that \( \langle A, \beta \rangle \models \Gamma \). By applying Definitions 4.1.14 and 4.1.11 we infer that \( tA,\beta = r_iA,\beta \in A^u \), \( s_iA,\beta \in A^u \), and that \( s_iA,\beta = s_iA,\beta \) if \( tA,\beta = r_iA,\beta \), for all \( j \neq i \). By applying Definition 4.1.11 again, we deduce from these facts that \( tA,\beta \in \text{DOM} \ (f[r_1\mapsto s_1, \ldots, r_i\mapsto s_i, \ldots, r_m\mapsto s_m]A,\beta) \), and \( (f[r_1\mapsto s_1, \ldots, r_i\mapsto s_i, \ldots, r_m\mapsto s_m]t)A,\beta = s_iA,\beta \), from which the soundness of this rule follows.

- **Xf3** We have to show that:

\[ \Gamma \models t = r_i \quad \& \quad \Gamma \models \neg \downarrow s_i \quad \Rightarrow \quad \Gamma \models \neg \downarrow f[\ldots, r_i\mapsto s_i, \ldots]t. \]

Assume that \( \Gamma \models t = r_i \), \( \Gamma \models \neg \downarrow s_i \), and \( \langle A, \beta \rangle \models \Gamma \). Applying Definitions 4.1.14 and 4.1.11, we infer that \( s_iA,\beta = \infty \), that \( tA,\beta = r_iA,\beta \), and, consequently, that \( tA,\beta \notin \text{DOM} \ (f[\ldots, r_i\mapsto s_i, \ldots]A,\beta) \), from which the result follows.

The other cases are (also) straightforward, but are left to the reader.

The next theorem states the soundness of our Fitch-style deduction system for our version of \( E \)-logic. The proof is by induction on the length of derivations. In order to express this induction we need a numbering of the formulae. It is clear, however, from Definition 3.3.1 and the example derivations in Chapter 3, that such a numbering is always possible.

### 4.3.4 Theorem Soundness

*Let* \( \Gamma \subseteq \text{FORM} \) and \( \varphi \in \text{FORM} \), *then*

\[ \Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi. \]
Proof. Suppose that $\Gamma \vdash \varphi$, then by Definition 3.3.8 there exists a derivation $\Pi = \langle \Phi, \Psi \rangle$, such that $\Phi \subseteq \Gamma$ and $\varphi \in \Psi$. Suppose that the formulae (nested) in $\Pi$ are numbered $1, 2, \ldots, n$, where $n$ is called the length of $\Pi$. Moreover, suppose that $\varphi = \varphi_n$. This means that we suppose that $\varphi$ is the last line of $\Pi$. If this were not the case, we could cut all formulae below $\varphi_n$ from $\Pi$, and we would still have a valid derivation for $\Gamma \vdash \varphi$.

Now, for any formula $\varphi_i$, define $\Gamma_i$ to be the set of all active hypotheses of the unique proof figure in which $\varphi_i$ is contained (with respect to $\Pi$). So, letting $\Pi_1 = \langle \Phi_1, \Psi_1 \rangle, \ldots, \Pi_m = \langle \Phi_m, \Psi_m \rangle$ be such that $\varphi_i \in \Pi_m \in \ldots \in \Pi_1 = \Pi$ ($m \geq 1$) we have:

$$\Gamma_i = \Phi_1 \cup \ldots \cup \Phi_m.$$ 

Then, in order to prove soundness it is sufficient to show by induction on $i$ that for any $1 \leq i \leq n$:

$$\Gamma_i \models \varphi_i.$$ 

From this fact, soundness follows by taking $i = n$, for then $\varphi_n = \varphi$ and $\Gamma_n = \Phi$ by definition of $\Gamma_i$.

We will give a sketch of the induction proof, which is very straightforward.

1. $i = 1$ (base case).

   If $\Gamma_1 = \emptyset$, then $\varphi_1$ has to be an axiom. Applying Theorem 4.3.3 does the rest. If $\Gamma_1 \neq \emptyset$, then $\varphi_1 \in \Gamma_1$, in which case the result is trivially true.

2. $i = j + 1$ (induction case).

   The induction hypothesis is, that $\Gamma_k \models \varphi_k$ for all $1 \leq k \leq j$.

Depending on by which rule $\varphi_{j+1}$ has been derived, we need one of the corresponding cases of Theorem 4.3.3. We will give two examples. Assume that $\varphi_{j+1} \in \Pi_m = \langle \Phi_m, \Psi_m \rangle \in \ldots \in \Pi_1 = \langle \Phi_1, \Psi_1 \rangle = \Pi$ ($m \geq 1$).

- $\varphi_{j+1}$ was derived by applying $\textbf{Rei}$.

   In this case, there must be a $k < j + 1$ such that $\varphi_k \equiv \varphi_{j+1}$ and $\varphi_k$ is visible in $\Pi_m$. This implies that $\varphi_k \in \Phi_l$ for some $l$ such that $1 \leq l \leq m$, or $\varphi_k \in \Psi_l$ for some $l$ such that $1 \leq l < m$. It is evident that in both cases we have that $\Gamma_k \subseteq \Gamma_{j+1}$. Moreover, by the induction hypothesis, we have $\Gamma_k \models \varphi_k$. But then, by monotonicity, we also have $\Gamma_{j+1} \models \varphi_{j+1}$.

- $\varphi_{j+1} \equiv (\varphi_k \rightarrow \varphi_l)$ was derived by applying $\textbf{→I}$.

   Here, there must be a proof figure $\Pi' = \langle \Phi', \Psi' \rangle \in \Pi_m$, such that $\Pi'$ precedes $\varphi_{j+1}$ in $\Psi_m$, $\Phi' = \{ \varphi_k \}$, and $\varphi_l$ is the last line of $\Psi'$. It follows that $\Gamma_l = \Gamma_{j+1} \cup \{ \varphi_k \}$. Now, by the induction hypothesis we have $\Gamma_{j+1}, \varphi_k \models \varphi_l$, from which we conclude $\Gamma_{j+1} \models \varphi_k \rightarrow \varphi_l$, by applying Theorem 4.3.3.
4.3. Soundness and completeness

This concludes the proof of the soundness theorem.

We will now set out to prove the completeness of our deduction system. In fact we will prove the stronger result that every consistent set has a model, from which completeness almost directly follows. This is, in fact, the strategy found in most textbooks. Only in our case we prove the result for our version of $E$-logic, which differs from classical logic at several points: our logic deals with partial functions, extended function symbols and extended terms. The latter two features of the logic complicate the proof.

First we will prove the Lindenbaum theorem which says that every consistent set of formulae $\Gamma$ can be extended to a maximal consistent set $\bar{\Gamma}$ with witnesses. Then we will show how to construct a model for $\bar{\Gamma}$. The latter result is called the Henkin theorem. These two theorems imply that any consistent set has a model.

4.3.5 Definition Let $\Gamma \subseteq \text{Form}$.

i. $\Gamma$ is called consistent if it is not the case that $\Gamma \vdash \text{ff}$.

ii. $\Gamma$ is called maximal consistent if $\Gamma$ is consistent and if $\Gamma \cup \{\varphi\}$ is inconsistent whenever $\varphi \in \text{Form}$ and $\varphi \notin \Gamma$.

iii. $\Gamma$ contains witnesses if for any existential formula $\exists x^u \varphi \in \Gamma$ there exists a term $t \in \text{Xterm}_u$ such that $\{\downarrow t, \varphi[x^u:=t]\} \subseteq \Gamma$.

4.3.6 Remark As we are dealing with a logic of partial functions, our definition of a set containing witnesses is adapted to this fact. Whereas in classical logic it is sufficient to demand that $\varphi[x^u:=t] \in \Gamma$ for some term $t$, if $\exists x^u \varphi \in \Gamma$, we also need in our case that this term $t$ actually is denoting.

4.3.7 Proposition Let $\Gamma \subseteq \text{Form}$ be a maximal consistent set and let $\varphi, \psi \in \text{Form}$, then:

i. If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.

ii. Either $\varphi \in \Gamma$, or $\neg \varphi \in \Gamma$.

iii. $(\varphi \lor \psi) \in \Gamma$ if and only if $\varphi \in \Gamma$ or $\psi \in \Gamma$.

iv. If $(\varphi \rightarrow \psi) \in \Gamma$ and $\varphi \in \Gamma$, then $\psi \in \Gamma$.

v. If, in addition, $\Gamma$ contains witnesses, then $\exists x^u \varphi \in \Gamma$ if and only if $\{\downarrow y^u, \varphi[x^u:=y^u]\} \subseteq \Gamma$ for some variable $y^u$.

Proof The proof consists of a simple adaptation of the corresponding proof for classical logic.

4.3.8 Theorem Lindenbaum

If $\Gamma \subseteq \text{Form}$ is consistent then there exists a maximal consistent set $\bar{\Gamma}$ such that $\Gamma \subseteq \bar{\Gamma}$ and $\bar{\Gamma}$ contains witnesses.
Chapter 4. The semantics of $E$-logic

Proof. We will show how to construct the set $\overline{\Gamma}$ out of $\Gamma$. In the construction we need a countably infinite set $\{y_1^u, y_2^u, \ldots, y_m^u, \ldots\}$ of fresh variables for any sort $u \in \text{SORT}$. Without loss of generality, we will assume these fresh variables to exist. In case $\text{VAR}_u \setminus \text{FV}(\Gamma)$ would be finite for some $u \in \text{SORT}$, we would be able to enrich the language by extending the set $\text{VAR}_u$ with countably infinite new fresh variables.

Moreover, we will need the fact that all elements of $\text{FORM}$ can be enumerated: $\varphi_1, \varphi_2, \ldots, \varphi_n, \ldots$. We will assume the reader to be familiar with techniques to provide such an enumeration.

Now, define:

\[
\begin{align*}
\Gamma_0 &= \Gamma \\
\Gamma_n &= \begin{cases}
\Gamma_n, & \text{if } \Gamma_n \cup \{\varphi_n\} \text{ is inconsistent,} \\
\Gamma_n \cup \{\varphi_n\}, & \text{if this set is consistent and} \\
\Gamma_n \cup \{\varphi_n, \downarrow y_m^u, \psi[x^u := y_m^u]\}, & \text{if } \Gamma_n \cup \{\varphi_n\} \text{ is consistent,} \\
\end{cases} \\
\Gamma_{n+1} &= \begin{cases}
\Gamma_n, & \varphi_n \text{ is not of form } \exists x^u \psi, \\
\end{cases} \\
\overline{\Gamma} &= \bigcup_{n \in \mathbb{N}} \Gamma_n.
\end{align*}
\]

The set $\overline{\Gamma}$ defined in this way turns out to be a maximal consistent set containing witnesses such that $\Gamma \subseteq \overline{\Gamma}$. The proof of this fact can be obtained by slightly adapting the corresponding proof for classical logic. ■

4.3.9 Definition. A $\Sigma$-interpretation $\mathcal{I} = \langle \mathfrak{A}, \beta \rangle$ is called countable if all universes $\mathfrak{A}_u$ of $\mathfrak{A}$ have countably many elements.

4.3.10 Theorem Henkin

Let $\Gamma \subseteq \text{FORM}$ be a maximal consistent set containing witnesses, then there exists a countable model $\mathcal{I}_\Gamma$ of $\Gamma$ such that:

$\mathcal{I}_\Gamma \models \varphi \iff \varphi \in \Gamma$.

Proof. The construction of the model $\mathcal{I}_\Gamma$ will be presented in several steps. First we will define a relation $\sim$ between terms. This relation will be based on the strong equalities which are contained in $\Gamma$. It will turn out that this relation is a congruence relation with respect to extended function symbols and predicate symbols. Then we will define the model $\mathcal{I}_\Gamma$ and prove that it is a countable model of $\Gamma$.

The universes of $\mathcal{I}_\Gamma$ will consist of the equivalence classes generated by $\sim$ on denoting terms. The value $\infty$ will be represented by the equivalence class of $\bot$ for every $u \in \text{SORT}$. Of course, these equivalence classes will not become members of any universe of $\mathcal{I}_\Gamma$, since they represent the 'undefined values'.
4.3. Soundness and completeness

Definition of the relation $\sim$ between terms.

Let $s, t \in \text{XTERM}_u$ for some $u \in \text{SORT}$, then define:

$$s \sim t \iff s \approx t \in \Gamma.$$  

Note that we use strong equality (see Definition 3.3.12) here. The reason for this is that we want that $\sim$ also determines an equivalence class of $\bot$.

In the following, if $s, t \in \text{XTERM}^w$, we will write $s \sim t$ as an abbreviation of $s_1 \sim t_1$, and $\ldots$, and $s_n \sim t_n$.

Proof that $\sim$ is an congruence relation.

The fact that $\sim$ is an equivalence relation almost directly follows from the fact that $\Gamma$ is maximal consistent. For example, since $\Gamma \vdash t = t$ for any term $t$, we must have $(t = t) \in \Gamma$ by Proposition 4.3.7, which implies reflexivity: $t \sim t$.

It has also to be shown that $\sim$ is a congruence relation with respect to extended function symbols and predicate symbols. What this means is the following. Let $s \sim t$ for $s, t \in \text{XTERM}^w$, then:

$$fs \sim ft,$$
$$Ps \in \Gamma \iff Pt \in \Gamma,$$

for any extended function symbol $f$ and predicate symbol $P$. The reader can easily verify that these properties hold, using Theorems 3.3.16 and 3.3.17, and Proposition 4.3.7.

Definition of $I_\Gamma$.

Given the equivalence relation $\sim$, we will now define the model $I_\Gamma = (\mathfrak{A}, \beta)$. The idea is, to use the equivalence classes generated by $\sim$ as the elements of the universes of $I_\Gamma$. So, if $t \in \text{XTERM}_u$, then the equivalence class of $t$ with respect to $\sim$, notation $[t]$, will be an element of $\mathfrak{A}_u$. We have to be cautious, however. Not all terms $t$ will be denoting. If, for example, $\neg t \in \Gamma$, then $t \in [\bot]$, since in that case $t \sim \bot$. Of course we don’t want that $[\bot] \in \mathfrak{A}_u$. What we do want is that $[\bot]$ is the ‘undefined value’ $\infty$ for any universe $\mathfrak{A}_u$.

This motivates the following definition of $I_\Gamma$. Let $u \in \text{SORT}$, $c \in \text{CON}$, $f \in \text{FUN}_{w, u}$, and $P \in \text{PRED}_{w}$. Moreover, let $[t]$ be an abbreviation of $([t_1], \ldots, [t_n])$, if $t = (t_1, \ldots, t_n) \in \text{XTERM}^w$. Then $I_\Gamma = (\mathfrak{A}, \beta)$ is defined by:

$$\mathfrak{A}_u = \{[t] \mid t \in \text{XTERM}_u \land \downarrow t \in \Gamma\},$$
$$c^\mathfrak{A} = \begin{cases} 
\{(\emptyset, [c])\} & \text{if } \downarrow c \in \Gamma, \\
\emptyset & \text{otherwise,} 
\end{cases}$$
$$\text{DOM}(f^\mathfrak{A}) = \{[t] \mid t \in \text{XTERM}^w \land \downarrow ft \in \Gamma\},$$
$$f^\mathfrak{A}[t] = [ft],$$
$$P^\mathfrak{A} = \{[t] \mid t \in \text{XTERM}^w \land \downarrow Pt \in \Gamma\},$$
$$\text{DOM}(\beta^u) = \{x^u \mid x^u \in \text{VAR}_u \land \downarrow x^u \in \Gamma\},$$
$$\beta^u(x^u) = [x^u].$$
Although it is not really necessary to specify the value of $\infty$ for each universe $\mathcal{A}_u$, we stipulate that:

$$\infty_{\mathcal{A}_u} = [\bot_u].$$

The advantage is that we now can prove (see below) that for any term $t$ we have $t^{I_T} = [t]$, even for those terms for which $\neg\bot t \in \Gamma$.

Note that the definitions of $f^{\mathcal{A}}$ and $P^{\mathcal{A}}$ are independent of the choice of the representatives of the equivalence classes, as $\sim$ is a congruence relation.

**Proof that $I_T$ is a countable model of $\Gamma$.**

It is easy to see that $I_T$ is countable, since for any $u \in \text{SORT}$ the set $X\text{TERM}_u$ is countable. So, we will concentrate on the important part: showing that $I_T$ is a model of $\Gamma$. We will prove three things, which are needed:

1. $t^{I_T} = [t]$ for all $t \in X\text{TERM}_u$.
2. $[t] \in \text{DOM}(f^{I_T})$ if and only if $\bot ft \in \Gamma$.
3. $I_T \models \varphi$ if and only if $\varphi \in \Gamma$.

The proof is by mutual induction on the structure of extended terms $t \in X\text{TERM}_u$ and extended function symbols $f \in X\text{FUN}_{w,u}$, and by ordinary induction on formulae $\varphi \in \text{FORM}$, respectively. We will consider the most interesting cases:

- $t \equiv \bot$. By definition of $\infty$, we have $\bot^{I_T} = \infty = [\bot]$.

- $t \equiv x^u$. If $x^u \in \text{DOM}(\beta)$, then $(x^u)^{I_T} = \beta(x^u) = [x^u]$ by definition of $I_T$. Otherwise, $(x^u)^{I_T} = \infty = [\bot] = [x^u]$, since in that case $\bot x^u \notin \Gamma$, which, by Proposition 4.3.7, implies that $\neg\bot x^u \in \Gamma$ and, therefore, $x^u \sim \bot$.

- $t \equiv c$. This case is similar to the preceding one.

- $t \equiv gt$ for some $g \in \text{FUN}_{w,u}$. According to the induction hypothesis we have $t^{I_T} = [t]$. Suppose that $[t] \in \text{DOM}(g^{I_T})$. Then, by definition, $(gt)^{I_T} = g^{I_T} t^{I_T} = g^\mathcal{A}[t] = [gt]$. If, on the other hand, $[t] \notin \text{DOM}(g^{I_T})$, then $(gt)^{I_T} = \infty = [\bot]$. But in that case, we also have $\neg\bot gt \in \Gamma$, which implies $gt \sim \bot$.

Note, that $[t] \in \text{DOM}(g^{I_T})$ if and only if $\bot gt \in \Gamma$, by definition of $g^\mathcal{A}$. (Remember that the choice of the representatives of the equivalence classes in $[t]$ is immaterial, since $\sim$ is a congruence relation.)

- $t \equiv g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t$ for some $g \in X\text{FUN}_{w,u}$. In this case we may apply the induction hypothesis to $t$, $g$, $gt$, $r_i$, and $s_i$ (for $1 \leq i \leq m$). We distinguish three possibilities:
4.3. Soundness and completeness

\(- t^\mathcal{T}_\Gamma = [t] \in \text{DOM} \left( g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\mathcal{T}_\Gamma} \right), \) and \([t] = t^\mathcal{T}_\Gamma = r_i^{\mathcal{T}_\Gamma} = [r_i] \) for some \(i\) such that \(1 \leq i \leq m\). This can only be the case if condition \(C(i)\) of Definition 4.1.11 holds.

Now, we have:

\[
\begin{align*}
(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t)^{\mathcal{T}_\Gamma} &= g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\mathcal{T}_\Gamma} t^{\mathcal{T}_\Gamma} \\
&= g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\mathcal{T}_\Gamma} [t] \\
&= s_i^{\mathcal{T}_\Gamma} \\
&= [s_i].
\end{align*}
\]

On the other hand, since condition \(C(i)\) and the induction hypothesis hold, we have that \([t] = [r_i] \in \mathcal{A}^w\) and \([s_i] \in \mathcal{A}_u\). Moreover, we have \([s_i] = [s_j]\) for all \(j \neq i\) such that \([r_i] = [r_j]\) and \(1 \leq j \leq m\). From these facts and from the fact that \(\Gamma\) is maximal consistent, we derive that \(\downarrow r_i \in \Gamma, \downarrow s_i \in \Gamma,\) and \((s_i = s_j) \in \Gamma\) for all \(j \neq i\) such that \((r_i = r_j) \in \Gamma\). But this implies that:

\[\Gamma \vdash g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = s_i.\]

So, by Proposition 4.3.7:

\[(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = s_i) \in \Gamma,\]

which means that:

\[g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = [s_i].\]

Combining this with what we found above, we conclude that in this case \(t^\mathcal{T}_\Gamma = [t]\).

From the fact that \(\Gamma \vdash g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = s_i\) and by applying Proposition 4.3.7, we infer that \(\downarrow g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t \in \Gamma,\) which we also had to prove.

\(- t^\mathcal{T}_\Gamma = [t] \in \text{DOM} \left( g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\mathcal{T}_\Gamma} \right), \) and \([t] = t^\mathcal{T}_\Gamma \neq r_i^{\mathcal{T}_\Gamma} = [r_i] \) for all \(i\) such that \(1 \leq i \leq m\). This implies that \([t] \in \text{DOM} \left( g^{\mathcal{T}_\Gamma} \right).\)

Now,

\[
\begin{align*}
(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t)^{\mathcal{T}_\Gamma} &= g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^{\mathcal{T}_\Gamma} t^{\mathcal{T}_\Gamma} \\
&= g^{\mathcal{T}_\Gamma} t^{\mathcal{T}_\Gamma} \\
&= [gt].
\end{align*}
\]

On the other hand, by the induction hypothesis for \(g\) we have that \([t] \in \text{DOM} \left( g^{\mathcal{T}_\Gamma} \right)\) if and only if \(\downarrow gt \in \Gamma\). Since we just have seen that \([t] \in \text{DOM} \left( g^{\mathcal{T}_\Gamma} \right),\) we infer that \(\downarrow gt \in \Gamma\). Taking this together with
the fact that \([t] \neq [r_i]\) for all \(i\) such that \(1 \leq i \leq m\), we deduce, applying Proposition 4.3.7:

\[ \Gamma \vdash g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = gt, \]

from which we derive:

\[ (g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = gt) \in \Gamma, \]

which implies:

\[ [g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t] = [gt]. \]

Combining this with what we found above, we conclude that also in this case we have \(t^\mathcal{I}_\Gamma = [t]\).

Again in this case, from \(\Gamma \vdash g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t = gt\) and Proposition 4.3.7, we infer that \(\downarrow g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t \in \Gamma\), which we also had to prove.

\[ \uparrow t^\mathcal{I}_\Gamma = [t] \notin \text{Dom} (g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^\mathcal{I}_\Gamma). \]

There could be various reasons for this, but we sure have:

\[ (g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t)^{\mathcal{I}_\Gamma} = \infty \]

\[ = [\bot]. \]

The reader can verify that it is possible to derive that:

\[ \neg \downarrow g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t \in \Gamma, \]

which, by Proposition 4.3.7, implies that:

\[ [g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t] = [\bot], \]

and

\[ \downarrow g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t \notin \Gamma, \]

both results we had to prove.

Combining the results of the analysis of the three cases above yields:

\[ (g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t)^{\mathcal{I}_\Gamma} = [g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t], \]

and

\[ [t] \in \text{Dom} (g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]^\mathcal{I}_\Gamma) \]

\[ \iff \downarrow g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]t \in \Gamma. \]

\[ \bullet \ \varphi \equiv \downarrow t \text{ for some } t \in \text{XTERM}_u. \]

Using \(t^\mathcal{I}_\Gamma = [t]\), we derive:

\[ \mathcal{I}_\Gamma \models \downarrow t \iff t^\mathcal{I}_\Gamma \in \mathcal{A}_u \]

\[ \iff [t] \in \mathcal{A}_u \]

\[ \iff \downarrow t \in \Gamma. \]
4.3. Soundness and completeness

- \( \varphi \equiv P t \). Again using \( t^\Gamma = [t] \), we derive:
  \[
  I_\Gamma \models P t \iff t^\Gamma \in P^\alpha \\
  \iff [t] \in \{ [s] \mid P s \in \Gamma \} \\
  \iff P t \in \Gamma.
  \]

- \( \varphi \equiv \neg \psi \). Using the induction hypothesis for \( \psi \), and Proposition 4.3.7, we derive:
  \[
  I_\Gamma \models \neg \psi \iff \sim I_\Gamma \models \psi \\
  \iff \psi \notin \Gamma \\
  \iff \neg \psi \in \Gamma.
  \]

- \( \varphi \equiv (\psi \vee \chi) \). Again using the induction hypothesis and Proposition 4.3.7, we derive:
  \[
  I_\Gamma \models \psi \vee \chi \iff I_\Gamma \models \psi \vee I_\Gamma \models \chi \\
  \iff \psi \in \Gamma \vee \chi \in \Gamma \\
  \iff (\psi \vee \chi) \in \Gamma.
  \]

- \( \varphi \equiv \exists x^u \psi \). Using the same facts and Theorem 4.2.17:
  \[
  I_\Gamma \models \exists x^u \psi \iff \exists [t] \in A_u \cdot (A, \beta[x^u \mapsto [t]]) \models \psi \\
  \iff \exists [t] \in A_u \cdot (A, \beta[x^u \mapsto t^\Gamma]) \models \psi \\
  \iff \exists t \in XTERM_u \cdot I_\Gamma \models \psi[x^u := t] \wedge \downarrow t \\
  \iff \exists t \in XTERM_u \cdot \{ \downarrow t, \psi[x^u := t] \} \subseteq \Gamma \\
  \iff \exists x^u \psi \in \Gamma.
  \]

Since the other cases are left to the reader, this concludes the proof of the Henkin theorem. 

4.3.11 Theorem If \( \Gamma \subseteq \text{FORM} \) is consistent, then \( \Gamma \) has a model.

Proof Suppose that \( \Gamma \) is consistent. Then, by Theorem 4.3.8 (Lindenbaum), there exists a maximal consistent set \( \bar{\Gamma} \) containing witnesses such that \( \Gamma \subseteq \bar{\Gamma} \). By Theorem 4.3.10 (Henkin) \( \bar{\Gamma} \) has a model \( I_{\bar{\Gamma}} \). It follows that \( I_{\bar{\Gamma}} \) is also a model of \( \Gamma \).

4.3.12 Theorem Completeness

Let \( \Gamma \subseteq \text{FORM} \) and \( \varphi \in \text{FORM} \), then:

\[
\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi.
\]
Chapter 4. The semantics of E-logic

Proof. Suppose that $\Gamma \models \varphi$. Then $\Gamma \cup \{\neg \varphi\}$ has no model. Using Theorem 4.3.11, we infer that $\Gamma \cup \{\neg \varphi\}$ is inconsistent. In that case, by Definitions 4.3.5 and 3.3.8, there exists a finite $\Gamma' \subseteq \Gamma$ such that $\Gamma' \cup \{\neg \varphi\}$ is inconsistent. But this implies that $\Gamma' \not\models \varphi$, by constructing a derivation using the rules $-I$ and $-E$. We conclude that $\Gamma \not\models \varphi$. □

4.3.13 Theorem Compactness

Let $\Gamma \subseteq \text{FORM}$. Then $\Gamma$ has a model if and only if every finite subset of $\Gamma$ has a model.

Proof. Using Theorem 4.3.11 and the definition of consistency (4.3.5) and derivability (3.3.8), we deduce:

\[
\begin{array}{c}
\Gamma \text{ has a model} \\
\iff \\
\text{every finite subset of } \Gamma \text{ is consistent} \\
\iff \\
\text{every finite subset of } \Gamma \text{ has a model},
\end{array}
\]

which concludes the proof. □

4.4 Standard E-logic

A variation of E-logic, called standard E-logic, can be obtained by fixing the interpretation of a part of a signature $\Sigma$. That part would be interpreted by the same mathematical structure in any $\Sigma$-interpretation. We will call this kind of interpretations standard $\Sigma$-interpretations. Standard interpretations are used in situations of mathematical discourse, such as reasoning about evolving algebras. In these situations one wants to make use of the properties of the fixed interpretation.

An example of a signature for which standard $\Sigma$-interpretations are likely to be used, is a signature $\Sigma$ containing the sort symbol $N$, the function symbols $\text{succ} \in \text{FUN}_{N,N}$ and $\text{plus} \in \text{FUN}_{N,N,N}$, and an individual constant $\text{zero} \in \text{CON}_N$. In any standard $\Sigma$-interpretation these symbols would be interpreted by the standard structure consisting of the universe $\mathbb{N}$ (the set of natural numbers), the function $S : \mathbb{N} \to \mathbb{N}$ (the successor function), the function $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (addition), and $0 : \mathbb{N}$ (zero).

Other examples involve structures which contain universes for character strings or derivation trees, together with various functions on those universes. Which universes and which functions are in the standard part of a signature is solely depending on the application domain where the logic is used.

In the literature standard logics which deal with natural numbers—in the sense that quantification over them is possible—are called $\omega$-logics (see e.g. [Bar85]). Due to Gödel's incompleteness theorem it is of course impossible to have a recursive axiomatization for these logics. As we will see, incompleteness can be repaired by taking as additional axioms all sentences which are true in the fixed structure, and by extending the notion of standard interpretation.
4.4. Standard E-logic

to the notion of semistandard interpretation. In the latter the standard part of a signature is interpreted by a structure which makes the same sentences true as the standard structure. In model theory these structures are called elementary equivalent.

4.4.1 Definition Let \( \Sigma_1 \) and \( \Sigma_2 \) be signatures. Then \( \Sigma_2 \) is a subsignature of \( \Sigma_1 \), notation \( \Sigma_2 \subseteq \Sigma_1 \), if \( \text{SORT}(\Sigma_2) \subseteq \text{SORT}(\Sigma_1) \), \( \text{SYM}(\Sigma_2) \subseteq \text{SYM}(\Sigma_1) \), and \( \sigma(\Sigma_2)(f) = \sigma(\Sigma_1)(f) \) for all \( f \in \text{SYM}(\Sigma_2) \).

4.4.2 Definition Let \( \mathfrak{A} \) be a \( \Sigma \)-structure and let \( \Sigma' \) be a subsignature of \( \Sigma \). Then the reduct of \( \mathfrak{A} \) with respect to \( \Sigma' \), notation \( \mathfrak{A} \upharpoonright \Sigma' \), is the \( \Sigma' \)-structure \( \langle B, G \rangle \) such that:

\[
B = \{ \mathfrak{A}_u \mid u \in \text{SORT}(\Sigma') \},
\]

\[
G = \{ f^\mathfrak{A} \mid f \in \text{SYM}(\Sigma') \}.
\]

4.4.3 Definition Let \( \mathcal{I} = (\mathfrak{A}, \beta) \) be a \( \Sigma \)-interpretation and let \( \Sigma' \) be a subsignature of \( \Sigma \). Then the reduct of \( \mathcal{I} \) with respect to \( \Sigma' \), notation \( \mathcal{I} \upharpoonright \Sigma' \), is the \( \Sigma' \)-interpretation \( \langle \mathfrak{A} \upharpoonright \Sigma', \gamma \rangle \), where:

\[
\gamma = \{ \beta^u \mid u \in \text{SORT}(\Sigma') \}.
\]

4.4.4 Definition Let \( \mathfrak{A} \) and \( \mathfrak{B} \) be \( \Sigma \)-interpretations. Then \( \mathfrak{A} \) and \( \mathfrak{B} \) are called elementary equivalent, notation \( \mathfrak{A} \equiv \mathfrak{B} \), if for all \( \varphi \in \text{SENT}(\Sigma) \):

\[
\mathfrak{A} \models \varphi \iff \mathfrak{B} \models \varphi.
\]

4.4.5 Definition Let \( \Sigma' \) be a subsignature of \( \Sigma \). Moreover, let \( \mathfrak{B} \) be a fixed \( \Sigma' \)-structure. Then \( \mathfrak{A} \) is called a standard \( \Sigma \)-structure with respect to \( \Sigma' \) and \( \mathfrak{B} \) if \( \mathfrak{A} \upharpoonright \Sigma' = \mathfrak{B} \). If \( \mathcal{I} = (\mathfrak{A}, \beta) \) is a \( \Sigma \)-interpretation, then \( \mathcal{I} \) is a standard \( \Sigma \)-interpretation with respect to \( \Sigma' \) and \( \mathfrak{B} \) if \( \mathfrak{A} \upharpoonright \Sigma' = \mathfrak{B} \).

4.4.6 Definition Let \( \Sigma' \) be a subsignature of \( \Sigma \). Moreover, let \( \mathfrak{B} \) be a fixed \( \Sigma' \)-structure. Then \( \mathfrak{A} \) is called a semistandard \( \Sigma \)-structure with respect to \( \Sigma' \) and \( \mathfrak{B} \) if \( \mathfrak{A} \upharpoonright \Sigma' \equiv \mathfrak{B} \). If \( \mathcal{I} = (\mathfrak{A}, \beta) \) is a \( \Sigma \)-interpretation, then \( \mathcal{I} \) is a semistandard \( \Sigma \)-interpretation with respect to \( \Sigma' \) and \( \mathfrak{B} \) if \( \mathfrak{A} \upharpoonright \Sigma' \equiv \mathfrak{B} \).

Note that standard structures and interpretations are also semistandard structures and interpretations, respectively. The converse need, of course, not to be true. For instance, it is well-known that structures exist which are elementary equivalent to the standard model of Peano arithmetic, but which are not isomorphic to it (see e.g. [CK90]). Although we call these structures 'semistandard', they are usually called nonstandard.

4.4.7 Convention If \( \Sigma' \) is a subsignature of \( \Sigma \), and \( \mathfrak{A} \) is a (semi)standard \( \Sigma \)-structure with respect to \( \Sigma' \) and \( \mathfrak{B} \), then we will call \( \Sigma' \) and \( \mathfrak{B} \) the standard parts of \( \Sigma \) and \( \mathfrak{A} \), respectively. We will also say say that \( \mathfrak{B} \) is the standard structure for \( \Sigma' \). Likewise for interpretations.
The notions of satisfaction, truth, validity and logical entailment for standard $E$-logic have almost the same definitions as in ordinary $E$-logic. The difference is that these notions are now defined using standard interpretations. For semistandard $E$-logic these notions are defined in terms of semistandard interpretations.

4.4.8 Definition Let $\Gamma \subseteq \text{FORM}$ and $\varphi \in \text{FORM}$ w.r.t. a language with signature $\Sigma$. Moreover, let $\Sigma'$ be the standard part of $\Sigma$ and let $\mathfrak{B}$ be the standard structure for $\Sigma'$.

i. $\varphi$ is satisfied by a standard $\Sigma$-interpretation $\mathcal{I}$ (with respect to $\Sigma'$ and $\mathfrak{B}$), if and only if $\varphi^\mathcal{I} = 1$. Notation: $\mathcal{I} \models_{\mathfrak{B}} \varphi$.

ii. $\Gamma$ is satisfied by a standard $\Sigma$-interpretation $\mathcal{I}$, if and only if $\mathcal{I} \models_{\mathfrak{B}} \psi$ for all $\psi \in \Gamma$. Notation: $\mathcal{I} \models_{\mathfrak{B}} \Gamma$.

iii. $\varphi$ is true in a standard $\Sigma$-structure $\mathfrak{A}$, if and only if $\langle \mathfrak{A}, \beta \rangle \models_{\mathfrak{B}} \varphi$ for all variable assignments $\beta$. Notation: $\mathfrak{A} \models_{\mathfrak{B}} \varphi$.

iv. $\varphi$ is valid or logically true, if and only if $\mathfrak{A} \models_{\mathfrak{B}} \varphi$ for all standard $\Sigma$-structures $\mathfrak{A}$. Notation: $\models_{\mathfrak{B}} \varphi$.

v. $\Gamma$ logically entails $\varphi$, or, equivalently, $\varphi$ is a logical consequence of $\Gamma$, if and only if $\mathcal{I} \models_{\mathfrak{B}} \Gamma$ implies $\mathcal{I} \models_{\mathfrak{B}} \varphi$ for all standard $\Sigma$-interpretations $\mathcal{I}$. Notation: $\Gamma \models_{\mathfrak{B}} \varphi$.

4.4.9 Definition The relation $\models^*_\mathfrak{B}$ is defined by replacing 'standard' by 'semistandard' and $\models_{\mathfrak{B}}$ by $\models^*_\mathfrak{B}$ everywhere in Definition 4.4.8.

To be capable of making use of the power of standard parts, we adapt the deduction system for the corresponding standard logic. This will be done by adding as axioms all formulae valid in the standard part of an interpretation.

4.4.10 Definition Let $\Sigma'$ be the standard part of a signature $\Sigma$, and let $\mathfrak{B}$ the standard structure for $\Sigma'$. Then the Fitch-style natural deduction system for the standard $E$-logic over signature $\Sigma$ consists of all axioms and deduction rules introduced in Definition 3.3.5, and all axioms belonging to the set $\text{AXIOM}(\mathfrak{B})$, which is defined as follows:

$$\text{AXIOM}(\mathfrak{B}) = \{ \varphi \in \text{SENT}(\Sigma') \mid \mathfrak{B} \models \varphi \}.$$  

The resulting deduction relation will be written as $\models_{\mathfrak{B}}$.

Model theorists call $\text{AXIOM}(\mathfrak{B})$ the theory of $\mathfrak{B}$. It turns out that $\models_{\mathfrak{B}}$ is sound and complete with respect to $\models^*_\mathfrak{B}$, but only sound with respect to $\models_{\mathfrak{B}}$. We will firstly prove that all models of $\text{AXIOM}(\mathfrak{B})$ have a standard part which is elementary equivalent to $\mathfrak{B}$. 
4.4.11 Proposition Let \( \Sigma' \) be the standard part of \( \Sigma \), and let \( \mathcal{B} \) be the standard structure for \( \Sigma' \). Moreover, let \( \mathcal{I} = (\mathcal{A}, \beta) \) be a \( \Sigma \)-interpretation such that \( \mathcal{I} \models \text{Axiom}(\mathcal{B}) \). Then \( \mathcal{A} \models \Sigma' \equiv \mathcal{B} \).

Proof Let \( \varphi \in \text{Sent}(\Sigma') \) and suppose that \( \mathcal{I} \models \text{Axiom}(\mathcal{B}) \).

Assume that \( \mathcal{B} \models \varphi \). Then, by Definition 4.4.10, \( \varphi \in \text{Axiom}(\mathcal{B}) \). Since \( \mathcal{I} \models \text{Axiom}(\mathcal{B}) \) and \( \varphi \in \text{Sent}(\Sigma') \) we infer, by applying Theorem 4.1.17, that \( \mathcal{A} \models \Sigma' \models \varphi \).

Conversely, assume that \( \mathcal{A} \models \Sigma' \models \varphi \). We have to show that \( \mathcal{B} \models \neg \varphi \). If this were not the case, then \( \mathcal{B} \models \neg \varphi \) and, consequently, \( \neg \varphi \in \text{Axiom}(\mathcal{B}) \). In contradiction to our assumption that \( \mathcal{A} \models \Sigma' \models \varphi \), this would imply that \( \mathcal{I} \models \neg \varphi \).

Combining our findings, we conclude that \( \mathcal{A} \models \Sigma' \equiv \mathcal{B} \).

4.4.12 Theorem Soundness and completeness semistandard E-logic

Let \( \varphi \in \text{Form} \) and \( \Gamma \subseteq \text{Form} \) with respect to some signature \( \Sigma \) with standard part \( \Sigma' \). Moreover, let \( \mathcal{B} \) be the standard structure for \( \Sigma' \). Then:

\[ \Gamma \vdash_\text{B} \varphi \iff \Gamma \models_\mathcal{B} \varphi. \]

Proof In the following argument, soundness and completeness of E-logic and Proposition 4.4.11 are used.

\[ \begin{array}{ll}
\Gamma \vdash_\text{B} \varphi & \iff \Gamma \cup \text{Axiom}(\mathcal{B}) \vdash \varphi \\
& \iff \Gamma \models \text{Axiom}(\mathcal{B}) \models \varphi \\
& \iff \forall \mathcal{I} \cdot (\mathcal{I} \models \Gamma \models \text{Axiom}(\mathcal{B}) \models \varphi) \\
& \iff \forall \mathcal{I} \cdot [\mathcal{I} \models \text{Axiom}(\mathcal{B}) \models (\mathcal{I} \models \Gamma \models \mathcal{I} \models \varphi)] \\
& \iff \Gamma \models_\mathcal{B} \varphi.
\end{array} \]

This concludes the proof.

4.4.13 Theorem Soundness standard E-logic

Let \( \varphi \in \text{Form} \) and \( \Gamma \subseteq \text{Form} \) with respect to some signature \( \Sigma \) with standard part \( \Sigma' \). Moreover, let \( \mathcal{B} \) be the standard structure for \( \Sigma' \). Then:

\[ \Gamma \vdash_\text{B} \varphi \Rightarrow \Gamma \models_\mathcal{B} \varphi. \]

Proof Since \( \Gamma \models_\mathcal{B} \varphi \) implies \( \Gamma \models_\mathcal{B} \varphi \), and by applying Theorem 4.4.12.

4.4.14 Theorem Incompleteness standard E-logic

There exist \( \varphi \in \text{Form} \) and \( \Gamma \subseteq \text{Form} \) with respect to some signature \( \Sigma \) and some standard interpretation \( \mathcal{B} \) for \( \Sigma \) such that:

\[ \Gamma \models_\mathcal{B} \varphi \land \neg \Gamma \vdash_\text{B} \varphi. \]
Chapter 4. The semantics of $E$-logic

Proof Assume that $\Sigma$ contains the sort symbol $N$, the constant zero and the function $\text{succ}$. Let the standard structure $\mathcal{B}$ contain the universe $\mathbb{N}$ of natural numbers, the constant $\text{zero}^\mathcal{B} = 0 \in \mathbb{N}$ and the successor function $\text{succ}^\mathcal{B} = S : \mathbb{N} \to \mathbb{N}$. Define $\varphi \equiv \text{ff}$ and:

$$\Gamma = \{ \lnot x, x \neq \text{zero}, x \neq \text{succ}(\text{zero}), x \neq \text{succ}(\text{succ}(\text{zero})), \ldots \}.$$ 

Then, by the Compactness Theorem (4.3.13), there exists a $\Sigma$-interpretation $\mathcal{J}$ which is a model of $\text{Axiom}(\mathcal{B}) \cup \Gamma$. In the model $\mathcal{J}$, the element denoted by $x$ has to be a nonstandard natural number.

Obviously, we do not have $\Gamma \models \text{ff}$. On the other hand, it must be the case that $\Gamma \models^\mathcal{B} \varphi$, since $\Gamma$ cannot be satisfied by any standard interpretation $\mathcal{I}$. Since:

$$\Gamma \models^\mathcal{B} \varphi \iff \forall \mathcal{I} \cdot [\mathcal{I} \models \text{Axiom}(\mathcal{B}) \Rightarrow (\mathcal{I} \models \Gamma \Rightarrow \mathcal{I} \models \varphi)],$$

and the right part of the equivalence clearly does not hold, the theorem directly follows.

By the soundness of standard $E$-logic, it is possible to use the natural deduction system of this logic to prove metatheorems of the form $\Gamma \models^\mathcal{B} \varphi$. In this way one would use the relation $\models^\mathcal{B}$. However, since the logic is not complete, we will almost exclusively employ the relation $\models$ in the remainder of this thesis.

Standard interpretations will be used in the next chapter to model states of evolving algebras. It seems very unnatural to allow for semistandard interpretations to play this role. This motivates the following convention.

4.4.15 Convention From now on we will only use standard $E$-logic with standard interpretations, and omit the qualification 'standard'. If from the context it is clear which standard part $\mathcal{B}$ is used, we will drop the subscript $\mathcal{B}$ in $\models^\mathcal{B}$ and $\models^\mathcal{B}$. 
Chapter 5

Syntax and semantics of Evolving Algebras

This chapter provides a formalization of Evolving Algebras (or simply EAs). We stress that we give a formalization, rather than the formalization, since the language of EAs is not a fixed formalism. For any purpose one has in mind new language features could be introduced. On the other hand practice will learn us which features will survive, and which won’t.

The kind of formalization we have chosen is based on many-sorted algebras. In this respect we differ from the approach by Gurevich (see e.g. [Gur91, Gur95]) which is based on one-sorted algebras. It is well-known, however, that many-sorted systems can be simulated by one-sorted systems, and vice versa. But, just like in the case of, for example, programming languages, languages which do have a type system, are commonly preferred to languages which do not have a type system. The reason for this preference is a pragmatic one. In Section 5.1 we will introduce a many-sorted language of EAs.

In logic it is customary to design a formal semantics whenever the syntax has been designed. For EAs this has not been done yet. Some of the designers of the Evolving Algebra approach have the conviction that EAs are their own semantics. Although in practice EAs are simple to understand, we believe that for foundational purposes a formal semantics is necessary. Therefore, we felt obliged to formulate a semantics for EAs. The kind of semantics that can be found in Section 5.2 is a structural operational one. Its adequacy and many other of its properties will be proven in Section 5.3. Another semantics, based on the axiomatic approach of Hoare and Floyd, will be presented in Chapter 6.

5.1 Syntax of Evolving Algebras

The underlying logic of EAs will, not surprisingly, be $E$-logic. As a consequence, the syntactical matters of $E$-logic, such as extended function symbols, extended terms, and substitution, directly apply to the language of EAs.

In the next definition the notion of a signature for a language of evolving algebras will be introduced. EAs are function-oriented, so there are no predicate
symbols. However, when the axiomatic semantics (see next chapter) comes into play, auxiliary functions and predicates can be useful. In that case the signature of an EA is thought of as being embedded in an $E$-logic signature. This notion of an embedded signature is a special case of the notion of subsignature in $E$-logic.

The idea of an EA signature is that there are two kinds of sort symbols. On the one hand there are the so-called static sort symbols $S\text{SORT}$ which are intended to denote fixed universes, and on the other hand we have the dynamic sort symbols which will refer to dynamic universes.

The static sort symbols come together with static function symbols, which denote fixed many-sorted functions over the fixed universes. The dynamic function symbols refer to functions which can have elements from static or dynamic universes as arguments. These are the functions which are updatable.

The static part of an EA signature is clearly related to the notion of standard part of a standard $E$-logic signature. As expected, this static part will be interpreted by the standard part of an interpretation.

Just like in $E$-logic, the function $\sigma$ assigns arities to the function symbols. Individual constants are considered to be nullary function symbols.

**5.1.1 Definition** A signature $\Sigma$ for a language of evolving algebras is an ordered quintuple

$$\Sigma = \langle S\text{SORT}, D\text{SORT}, SS\text{YM}, DS\text{YM}, \sigma \rangle,$$

such that:

1. $S\text{SORT}$, $D\text{SORT}$, $SS\text{YM}$ and $DS\text{YM}$ are sets of symbols satisfying the requirements $S\text{SORT} \cap D\text{SORT} = \emptyset$ and $SS\text{YM} \cap DS\text{YM} = \emptyset$,

2. $\sigma$ is a function $\sigma : SS\text{YM} \rightarrow S\text{SORT}^* \times S\text{SORT}$ where:

   - $SS\text{YM} = SS\text{YM} \cup DS\text{YM}$, and
   - $S\text{SORT} = S\text{SORT} \cup D\text{SORT}$,

and $\sigma$ satisfies the condition $\sigma(f) \in S\text{SORT}^* \times S\text{SORT}$ for all symbols $f \in SS\text{YM}$.

The elements of $S\text{SORT}$ or $D\text{SORT}$ are called static and dynamic sort symbols, respectively. The elements of $S\text{SORT}$ are just called sort symbols. Likewise, the elements of $SS\text{YM}$ or $DS\text{YM}$ are called static and dynamic symbols, whereas the elements of $SS\text{YM}$ are simply called symbols. $\sigma$ is the arity function.

**5.1.2 Convention** i. Elements $c$ of $SS\text{YM}$ or $DS\text{YM}$ for which $\sigma(c) = (\lambda, u)$ for some $u \in S\text{SORT}$ are called static and dynamic individual constants, respectively. The sets of static and dynamic individual constants with arity $(\lambda, u)$ are denoted by $S\text{CON}_u$ and $D\text{CON}_u$, respectively. The set $\text{CON}_u$ of all individual constants of arity $(\lambda, u)$ is defined by $\text{CON}_u = S\text{CON}_u \cup D\text{CON}_u$. 
5.1. Syntax of Evolving Algebras

ii. Elements $f$ of $\text{SSYM}$ or $\text{DSYM}$ for which $\sigma(c) = (w, u)$ for some $w \in \text{SORT}^+$ and $u \in \text{SORT}$ are called static and dynamic function symbols, respectively. The set of static and dynamic function symbols with arity $(w, u)$ are denoted by $\text{SFUN}_{w,u}$ and $\text{DFUN}_{w,u}$, respectively. The set $\text{FUN}_{w,u}$ of all function symbols of arity $(w, u)$ is defined by $\text{FUN}_{w,u} = \text{SFUN}_{w,u} \cup \text{DFUN}_{w,u}$.

iii. Regarding metavariables we will use the same conventions as in $E$-logic.

5.1.3 Definition Let $\Sigma_1$ and $\Sigma_2$ be signatures for EAs. Then $\Sigma_2$ is a subsignature of $\Sigma_1$, notation $\Sigma_2 \subseteq \Sigma_1$, if $\text{SSORT}(\Sigma_2) \subseteq \text{SSORT}(\Sigma_1)$, $\text{DSORT}(\Sigma_2) \subseteq \text{DSORT}(\Sigma_1)$, $\text{SSYM}(\Sigma_2) \subseteq \text{SSYM}(\Sigma_1)$, $\text{DSYM}(\Sigma_2) \subseteq \text{DSYM}(\Sigma_1)$, and $\sigma(\Sigma_2)(f) = \sigma(\Sigma_1)(f)$ for all $f \in \text{SYM}(\Sigma_2)$.

5.1.4 Definition Let $\Sigma_1$ be a signature for $E$-logic, and $\Sigma_2$ be a signature for $EA$. Then $\Sigma_2$ is said to be embedded in $\Sigma_1$, notation $\Sigma_2 \subseteq \Sigma_1$, if $\text{SORT}(\Sigma_2) \subseteq \text{SORT}(\Sigma_1)$, $\text{SYM}(\Sigma_2) \subseteq \text{SYM}(\Sigma_1)$, and $\sigma(\Sigma_2)(f) = \sigma(\Sigma_1)(f)$ for all $f \in \text{SYM}(\Sigma_2)$.

5.1.5 Definition Let $\Sigma = (\text{SSORT}, \text{DSORT}, \text{SSYM}, \text{DSYM}, \sigma)$ be a signature. Then the static part of $\Sigma$, notation $\Sigma^s$, is defined by:

$$\Sigma^s = (\text{SSORT}, \text{SSYM}, \sigma'),$$

where $\sigma'(f) = \sigma(f)$ for all $f \in \text{SSYM}$.

5.1.6 Remark It is easy to see that $\Sigma^s$ is an EA signature for any signature $\Sigma$. This follows from the requirement put on the arity function $\sigma$. In general, it is not possible to define a notion of a dynamic part in the same way: dynamic functions may have arguments from static universes.

5.1.7 Definition Associated with a signature $\Sigma$ with sorts $\text{SORT}$, a family of mutually disjoint, countably infinite sets $\text{VAR}_u$ of variables of sort $u$ for each $u \in \text{SORT}$ is given. The sets $\text{VAR}_u$ are disjoint from the set $\text{SYM}$.

5.1.8 Convention If $\Sigma_1$ is a signature for $E$-logic, and $\Sigma_2$ is a signature for $EA$ such that $\Sigma_2$ is embedded in $\Sigma_1$, then we will assume that $\text{VAR}_u(\Sigma_2) = \text{VAR}_u(\Sigma_1)$ for all $u \in \text{SORT}(\Sigma_2)$.

The sets $\text{TERM}_u$ of terms, the sets $\text{XFUN}_{w,u}$ of extended function symbols, the sets $\text{XTERM}_u$ of extended terms, and the set $\text{FORM}$ of formulae are defined in exactly the same way as in $E$-logic (see Section 3.1). This is also true for the sets of free variables, denoted by $\text{FV}(\_)$, occurring in extended function symbols, (extended) terms, and (sets of) formulae. Moreover, the notion of substitution and the notion of affecting substitution, are defined in the same way (see Section 3.2).
Chapter 5. Syntax and semantics of Evolving Algebras

Now we will define the notion of an Evolving Algebra over a signature \( \Sigma \). Recall from the informal introduction in Chapter 2 that an evolving algebra consists of some finite set of so-called rules. Rules are composed of conditions and updates. A rule can only be executed if its condition, also called its guard, evaluates to true in the current interpretation. Execution of a rule amounts to simultaneously performing the updates of the rule on the current interpretation. There are simple updates and extension updates.

A simple update changes the value of a term, or removes an element from a universe while making all necessary adjustments to restrict the interpretation of variables, constants and functions to the new universe. Updates of the first kind are called (local) function updates, and those of the second kind contraction updates. Not all terms can be subject to updates; the set of so-called updatable terms is defined below.

An extension update adds a new element to a designated universe and performs the updates occurring in the scope of that extension update. These nested updates involve the newly created element, which is locally bound to the so-called extension variable. In this way, extension updates provide some form of structuring.

5.1.9 Definition Let \( t \in \text{TERM}_u \), then \( t \) is called an updatable term if

1. \( t \equiv x^u \) for some \( x^u \in \text{VAR}_u \), or
2. \( t \equiv c \) for some \( c \in \text{DCON}_u \), or
3. \( t \equiv ft \) for some \( f \in \text{DFUN}_{w,u} \) and some vector \( t \in \text{TERM}^w \).

The set of updatable terms of sort \( u \in \text{SORT} \) is denoted by \( \text{UTERM}_u \).

5.1.10 Proposition \( \text{UTERM}_u \subseteq \text{TERM}_u \subseteq \text{XTERM}_u \) for all \( u \in \text{SORT} \).

5.1.11 Definition Let a signature \( \Sigma \) be given. The set \( \text{COND} \) of conditions is defined by:

1. \( tt, ff \in \text{COND} \),
2. \( t \in \text{TERM}_u \Rightarrow \downarrow t \in \text{COND} \),
3. \( s, t \in \text{TERM}_u \Rightarrow (s = t) \in \text{COND} \),
4. \( \varphi, \psi \in \text{COND} \Rightarrow \neg \varphi, (\varphi \land \psi), (\varphi \lor \psi) \in \text{COND} \),

Note that \( \text{COND} \subseteq \text{FORM} \).

5.1.12 Convention For any sets \( V \) and \( W \), the expression \( V \subseteq_{\text{fin}} W \) means that \( V \) is a finite subset of \( W \).

5.1.13 Definition Let a signature \( \Sigma \) be given. The set \( \text{UPD} \) of updates is recursively defined to be the smallest set such that:
5.1. Syntax of Evolving Algebras

1. \( u \in \text{SORT} \land s \in \text{UTERM}_u \land t \in \text{TERM}_u \Rightarrow s := t \in \text{UPD}, \)

2. \( u \in \text{DSORT} \land t \in \text{UTERM}_u \Rightarrow \text{rem } t : u \in \text{UPD}, \)

3. \( u \in \text{DSORT} \land x^u \in \text{VAR}_u \land U \subseteq \text{fin UPD} \)
   \( \Rightarrow \text{new } x^u : u \text{ with } U \in \text{UPD}, \)

Updates that are generated by the clauses 1 or 2 above, are called simple updates; those which are generated by clause 1 alone, are called (local) function updates, and those generated by clause 2 alone, are called contraction updates; the ones generated by 3 are called extension updates.

5.1.14 Definition Let a signature \( \Sigma \) be given. The set Rule of rules is the smallest set such that:

\[
\varphi \in \text{COND} \land U \subseteq \text{fin UPD} \land U \neq \emptyset \Rightarrow \text{if } \varphi \text{ then } U \in \text{RULE}. \]

The condition \( \varphi \) is also called the guard of the rule.

5.1.15 Definition An evolving algebra is a finite, non-empty set of rules, i.e. \( \mathcal{R} \) is an evolving algebra if \( \mathcal{R} \subseteq \text{fin RULE} \) and \( \mathcal{R} \neq \emptyset. \)

5.1.16 Convention

i. Variables ranging over elements of UPD are denoted by: \( U, V, \ldots. \)

ii. Variables ranging over finite subsets of UPD are denoted by: \( U, V, \ldots. \)

iii. Variables ranging over elements of RULE are denoted by: \( R, S, \ldots. \)

iv. Variables ranging over evolving algebras are denoted by: \( \mathcal{R}, S, \ldots. \)

The next definition concerns the notion of a free variable in an update, a rule, or an evolving algebra. Its definition is straightforward if one realizes that universe extensions new \( x^u : u \) with \( U \) have a binding effect on the free variables \( x^u \) occurring in the updates in \( U. \)

5.1.17 Definition If \( U \in \text{UPD} \) then the set \( \text{Fv}(U) \) of free variables in \( U \) is recursively defined by:

\[
\text{Fv}(s := t) = \text{Fv}(s) \cup \text{Fv}(t),
\]

\[
\text{Fv}(\text{rem } s : u) = \text{Fv}(s),
\]

\[
\text{Fv}(\text{new } x^u : u \text{ with } U) = \left( \bigcup_{U \in U} \text{Fv}(U) \right) \setminus \{x^u\}.
\]

If \( R \in \text{RULE} \) then \( \text{Fv}(R) \) is defined by:

\[
\text{Fv}(\text{if } \varphi \text{ then } U) = \text{Fv}(\varphi) \cup \bigcup_{U \in U} \text{Fv}(U),
\]
Chapter 5. Syntax and semantics of Evolving Algebras

If \( \mathcal{R} \) is an evolving algebra then \( \text{Fv}(\mathcal{R}) \) is defined by:

\[
\text{Fv}(\mathcal{R}) = \bigcup_{R \in \mathcal{R}} \text{Fv}(R).
\]

In this thesis we will only consider closed evolving algebras, that is evolving algebras \( \mathcal{R} \) such that \( \text{Fv}(\mathcal{R}) = \emptyset \). Allowing free variables can serve different purposes like bounded or unbounded parallelism and nondeterministic choice (see [Gur95]). We will not go into these possibilities here.

5.1.18 Example We will give an example of an evolving algebra \( \mathcal{R} \), such that \( \mathcal{R} \) computes the reverse of a character string. First we specify the signature \( \Sigma \) of \( \mathcal{R} \). In the specification of \( \Sigma \) we will use identifiers the meaning of which directly follows from their names. Furthermore, if \( \sigma(f) = (u_1 \ldots u_n, u) \) for some \( n > 0 \), we will write according to common mathematical practice \( f : u_1 \times \ldots \times u_n \rightarrow u \). If \( \sigma(c) = (\lambda, u) \), we will write \( c : u \).

Signature \( \Sigma \):
The signature \( \Sigma \) of \( \mathcal{R} \) is defined by the sets:

\[
\begin{align*}
\text{SSORT} &= \{ \text{Char}, \text{Char}^* \}, \\
\text{DSORT} &= \{ \text{Stack} \}, \\
\text{SSYM} &= \{ \text{head}, \text{tail}, \text{nil}, \text{append} \}, \\
\text{DSYM} &= \{ \text{input}, \text{output}, \text{bottom}, \text{top}, \text{next}, \text{content} \}.
\end{align*}
\]

The function \( \sigma \) is such that:

\[
\begin{align*}
\text{input} & : \text{Char}^*, \\
\text{output} & : \text{Char}^*, \\
\text{head} & : \text{Char}^* \rightarrow \text{Char}, \\
\text{tail} & : \text{Char}^* \rightarrow \text{Char}^*, \\
\text{nil} & : \text{Char}^*, \\
\text{append} & : \text{Char}^* \times \text{Char} \rightarrow \text{Char}^*, \\
\text{bottom} & : \text{Stack}, \\
\text{top} & : \text{Stack}, \\
\text{next} & : \text{Stack} \rightarrow \text{Stack}, \\
\text{content} & : \text{Stack} \rightarrow \text{Char}.
\end{align*}
\]

Note that the symbol * in the name \( \text{Char}^* \) is not a type constructor: the name \( \text{Char}^* \) is not compound.

The intended semantics of the elements of the signature is as follows: \( \text{Char} \) refers to some fixed set of characters, \( \text{Char}^* \) refers to the set of finite strings of elements of \( \text{Char} \), and \( \text{Stack} \) denotes a dynamic universe which will be used to model an auxiliary stack.
5.1. Syntax of Evolving Algebras

The functions head, tail and append are interpreted as the standard functions to compute the head and the tail of a character string, and the result of appending a character to a character string. The constant nil denotes the empty string.

The dynamic constants top and bottom are used to mark the top and the bottom of the auxiliary stack, whereas the function corresponding to next imposes a linked list like structure upon the elements of Stack. The function content is used to denote the contents of the elements of the stack.

Finally, the constants input and output refer to the input string to be reversed and the result of the reversion, respectively.

Evolving algebra \( \mathcal{R} \):

\[
\begin{align*}
&\text{if } \text{input} \neq \text{nil} \text{ then} \\
&\quad \text{new } x : \text{Stack with} \\
&\quad \quad \text{top} := x \\
&\quad \quad \text{next}(x) := \text{top} \\
&\quad \quad \text{content}(x) := \text{head}(\text{input}) \\
&\quad \text{input} := \text{tail}(\text{input}) \\
&\text{if } \text{input} = \text{nil} \land \text{top} \neq \text{bottom} \text{ then} \\
&\quad \text{output} := \text{append}(\text{output}, \text{content}(\text{top})) \\
&\quad \text{top} := \text{next}(\text{top}) \\
&\quad \text{rem top : Stack}
\end{align*}
\]

Note that we have suppressed the set brackets \{ and \} to delimit sets of updates. Instead of these we have used indentation to delimit the scope of extension updates. Also note that the variable \( x \) is used without superscript Stack: the sort of \( x \) obviously follows from the context.

In Chapter 6 of this thesis we will introduce a Hoare-style calculus for reasoning about EAs. The calculus enables the user to derive partial (and total) correctness formulae of the form:

\(
\{ \text{Precondition} \} \mathcal{R} \{ \text{Postcondition} \}.
\)

Here, \( \mathcal{R} \) refers to the execution of a run of the evolving algebra \( \mathcal{R} \) with respect to an initial state which is characterized by the formula Precondition. By a run we mean a series of states which starts in this initial state, and in which all successive states are the result of applying one of the applicable rules on the current state. A rule is applicable if its guard is true in the current state and if the updates of the rule are consistent with respect to the current state (this last condition will be extensively studied in the next section). The correctness formula expresses the property that Postcondition will hold in the final state of a run of \( \mathcal{R} \), provided that Precondition holds in the initial state of the run, and provided that the execution of \( \mathcal{R} \) halts.

With this calculus we will be able to prove that \( \mathcal{R} \) really computes the reverse output of the string input. In fact, we will be able to prove the correctness formula:
\{ \text{input} = \text{string} \land \text{output} = \text{nil} \land \text{top} = \text{bottom} \} \]

\[ \mathcal{R} \]
\[ \{ \text{length(output)} = \text{length(string)} \]
\[ \land \forall m : \mathbb{N} \cdot [m \geq 1 \land m \leq \text{length(string)} \]
\[ \rightarrow \text{at(output, m)} = \text{at(string, length(string) + 1 - m)} \} \}

In this formula, \text{string} denotes an auxiliary constant of sort \text{Char}^*\). Moreover, \text{N} is an auxiliary sort symbol denoting the natural numbers, and:

\[ \text{length} : \text{Char}^* \rightarrow \mathbb{N}, \]
\[ \text{at} : \text{Char}^* \times \mathbb{N} \rightarrow \text{Char}, \]

are auxiliary functions such that \text{length(string)} denotes the length of \text{string}, and \text{at(string, m)} the \text{m}th character of \text{string}.

5.1.19 Definition Let \( x^u, y^u \in \text{VAR}_u \). Then \( [x^u := y^u] : \text{UPD} \rightarrow \text{UPD} \) is called a \textbf{free variable renaming}. Its effect is recursively defined by:

\[
\begin{align*}
(s := t)[x^u := y^u] & = s[x^u := y^u] := t[x^u := y^u], \\
(\text{rem } t : v)[x^u := y^u] & = \text{rem } t[x^u := y^u] : v, \\
\text{(new } x^v : v \text{ with } U)[x^u := y^u] & = \begin{cases} 
\text{new } x^v : v \text{ with } U[x^u := y^u] & \text{if } x^u \neq x^v \text{ and } [y^u \neq x^v,} \\
& \text{or } x^u \notin \text{Fv}(U), \} \\
\text{new } z^v : v \text{ with } U[x^v := z^v][x^u := y^u] & \text{if } x^u \neq x^v, y^u \equiv x^v \text{ and} \\
& x^u \in \text{Fv}(U), \text{ where } z^v \\
\text{new } x^v : v \text{ with } U & \text{is the first fresh variable,} \\
& \text{otherwise.}
\end{cases}
\end{align*}
\]

With respect to the second case of the last equation, the variable \( z^v \) is called \textbf{fresh} if \( z^v \notin \text{Fv}(U) \cup \{y^u\} \).

If \( U = \{U_1, \ldots, U_m\}, \) then \( U[x^u := y^u] \) is defined by:

\[
U[x^u := y^u] = \{U_1[x^u := y^u], \ldots, U_m[x^u := y^u]\}.
\]

5.1.20 Remark With the help of the definition above, one can define the notion of a \textit{bound variable renaming}. If new \( x^u : u \) with \( U \in \text{UPD} \) and \( y^u \) is a fresh variable with respect to \( U \), i.e. \( y^u \notin \text{Fv}(U) \), then new \( y^u : u \) with \( U[x^u := y^u] \) is the result of a bound variable renaming of \( x^u \). We will prove in the next section that a bound variable renaming results in an operationally equivalent update (see Theorem 5.3.11).

5.1.21 Convention If \( x^u \in \text{VAR}_u \) and if the sort of \( x^u \) can be determined from the context, we will often write \( x \) instead of \( x^u \).
5.2 Operational semantics of Evolving Algebras

We will continue by describing the semantics of evolving algebras. The idea is that the execution of an evolving algebra induces a computation which is a sequence of states. Each next state in the sequence is the result of applying in parallel the updates of exactly one rule of the evolving algebra to the current state. The rule that is chosen to be executed is a rule whose guard is true in the current state. If there is more than one rule whose guard evaluates to true in the current state, one of them is chosen non-deterministically. If no guard is true, then the computation halts. We will be mainly interested in EAs in which at most one guard is true in each state in a computation. EAs with this property are called deterministic.

States will be interpretations over the signature of the evolving algebra, just like interpretations over signatures of $E$-logic. Because an interpretation for an EA signature only contains denotations of function symbols — recall that predicate symbols do not occur in EA signatures —, this kind of interpretation is called an algebra. This means that states are $\Sigma$-algebras over the signature $\Sigma$ of an EA. To be more precise: states will be pairs consisting of an algebra and a variable assignment.

5.2.1 Definition Let $\Sigma = (\text{SORT}, \text{DSORT}, \text{SYM}, \text{DSYM}, \sigma)$ be a given signature. A many-sorted partial algebra $\mathfrak{A}$ over $\Sigma$ is an ordered pair:

$$\mathfrak{A} = (A, F),$$

such that:

1. $A$ is a SORT-indexed family of, possibly empty, sets, i.e.

$$A = \{ \mathfrak{A}_u \mid u \in \text{SORT} \}.$$

The sets $\mathfrak{A}_u$ are called the universes of $\mathfrak{A}$.

2. $F$ is a $\sigma$-indexed family of partial functions on $A$, i.e.

$$F = \{ f^\mathfrak{A} \mid f \in \text{SYM} \}$$

where $f^\mathfrak{A} : \mathfrak{A}_w \rightarrow \mathfrak{A}_u$ whenever $\sigma(f) = (w, u)$.

A many-sorted partial algebra over $\Sigma$ will be called a $\Sigma$-algebra.

In the next definition we introduce standard $\Sigma$-algebras. These are, like standard $\Sigma$-structures in $E$-logic, $\Sigma$-algebras in which the static sort symbols and static function symbols have a standard, fixed, interpretation. For example, in a standard algebra over the signature of Example 5.1.18 the static sort symbol $\text{Char}^*$ will be interpreted as the set of character strings (over a fixed alphabet). Likewise, the function symbol $\text{append}$ will denote the append function for (character) strings.
Furthermore, in the same way we have done for $E$-logic, we will define the notion of a standard interpretation for EAs. Such an interpretation will be a pair consisting of an algebra and a variable assignment.

So, in a standard $\Sigma$-interpretation the denotations of the static symbols can be considered as fixed to their intended meaning. The choice of the interpretation depends on the application domain where the EA is used. Static functions give an enormous power to EAs. Since it is allowed to have non-computable functions as static functions, the power of EAs trivially exceeds the power of Turing machines. But if all static functions of an EA are computable, then this EA 'represents' a computable function.

The advantage of the possibility to choose static functions is that an EA can be tailored to its specific application domain. By choosing suitable static functions, the EA specification can be given a pragmatically right abstraction level in which all unimportant details are hidden in the functionality of the static functions.

Recall Definition 4.4.2 where the notion of a reduct was defined. This notion can be easily adapted to the case of a $\Sigma$-algebra $A$ and the static part $\Sigma^s$ of $\Sigma$. So, we will write $A \upharpoonright \Sigma^s$ for the reduct of $A$ with respect to $\Sigma^s$.

**5.2.2 Definition** Let $\Sigma$ be a signature with static part $\Sigma^s$, and let $A$ be a $\Sigma$-algebra. Moreover, let $B$ be a fixed $\Sigma^s$-algebra. Then $A$ is called a **standard $\Sigma$-algebra with respect to $B$**, if $A \upharpoonright \Sigma^s = B$.

**5.2.3 Definition** Let $A$ be a $\Sigma$-algebra, and let $\{\text{VAR}_u \mid u \in \text{SORT}\}$ be the family of sets of variables which is associated with $\Sigma$. A (variable) **assignment** $\beta$ (w.r.t. $\Sigma$ and $A$) is a $\text{SORT}$-indexed family of partial functions $\{\beta^u \mid u \in \text{SORT}\}$ such that $\beta^u : \text{VAR}_u \rightarrow A_u$ for all $u \in \text{SORT}$.

Note that variable assignments are partial functions. This means that variables are not necessarily denoting terms. Here we differ from [GR93] who use total variable assignments. This is also the reason we have adopted $E$-logic, rather than $E^+$-logic in which free variables are always denoting. We will come back to this point later in this chapter (see Example 5.2.42).

**5.2.4 Definition** A $\Sigma$-interpretation for a language with signature $\Sigma$ is an ordered pair $\langle A, \beta \rangle$ where $A$ is a $\Sigma$-algebra and $\beta$ is a variable assignment. If $A$ is a standard $\Sigma$-algebra with respect to $B$, then $A, \beta$ is called a **standard $\Sigma$-interpretation with respect to $B$**.

**5.2.5 Convention** Let $A$ be a standard $\Sigma$-algebra with respect to $B$. Then $B$ is called the **carrier** or **standard part** of $A$. We also say that $B$ is the **standard algebra** for $\Sigma^s$. Likewise for interpretations.

**5.2.6 Remark** Note that a standard $\Sigma$-interpretation for an EA is also a standard $\Sigma$-interpretation as defined for $E$-logic. This means that all of the machinery of the semantics of $E$-logic is at our disposal.
5.2.7 Convention If $\mathcal{I} = (\mathfrak{A}, \beta)$ is a $\Sigma$-interpretation, then $|\mathcal{I}|$ denotes the algebra $\mathfrak{A}$ in $\mathcal{I}$, i.e. $|\mathcal{I}| = \mathfrak{A}$.

By means of an interpretation terms and formulae can be interpreted, just like in $E$-logic. So, we will omit the corresponding definitions. The reader is referred to Chapter 4.

In defining satisfaction, truth, logical truth and logical entailment, we will restrict our attention to standard $\Sigma$-algebras. The effect of this choice is that, for example, the formulae $1 + 1 = 2$ and $\forall x : \text{Char}^*(\text{head}(\text{append}(\text{nil}, x)) = x)$ are considered logical truths. For convenience, we will give the appropriate definitions.

5.2.8 Definition Let $\Gamma \subseteq \text{Form}$ and $\varphi \in \text{Form}$ w.r.t. a language of EAs with signature $\Sigma$. Moreover, let $\mathfrak{B}$ be the standard algebra for $\Sigma^g$.

i. $\varphi$ is satisfied by a standard $\Sigma$-interpretation $\mathcal{I}$ (with respect to $\mathfrak{B}$), if and only if $\varphi^\mathcal{I} = 1$. Notation: $\mathcal{I} \models_{\mathfrak{B}} \varphi$.

ii. $\Gamma$ is satisfied by a standard $\Sigma$-interpretation $\mathcal{I}$, if and only if $\mathcal{I} \models_{\mathfrak{B}} \psi$ for all $\psi \in \Gamma$. Notation: $\mathcal{I} \models_{\mathfrak{B}} \Gamma$.

iii. $\varphi$ is true in a standard $\Sigma$-algebra $\mathfrak{A}$, if and only if $(\mathfrak{A}, \beta) \models_{\mathfrak{B}} \varphi$ for all variable assignments $\beta$. Notation: $\mathfrak{A} \models_{\mathfrak{B}} \varphi$.

iv. $\varphi$ is logically true, if and only if $\mathfrak{A} \models_{\mathfrak{B}} \varphi$ for all standard $\Sigma$-algebras $\mathfrak{A}$. Notation: $\models_{\mathfrak{B}} \varphi$.

v. $\Gamma$ logically entails $\varphi$, or, equivalently, $\varphi$ is a logical consequence of $\Gamma$, if and only if $\mathcal{I} \models_{\mathfrak{B}} \Gamma$ implies $\mathcal{I} \models_{\mathfrak{B}} \varphi$ for all standard $\Sigma$-interpretations $\mathcal{I}$. Notation: $\Gamma \models_{\mathfrak{B}} \varphi$.

5.2.9 Convention Since from the context it is always clear which $\mathfrak{B}$ is used as carrier, the subscript $\mathfrak{B}$ in $\models_{\mathfrak{B}}$ will be dropped. Moreover, unless explicitly stated otherwise, we will work with standard $\Sigma$-interpretations and drop the qualification 'standard'.

Before we can define the operational semantics of EAs, we have to specify when the updates of an EA rule can be executed on a given interpretation. If they can, we call them consistent with respect to that interpretation. The consistency condition we will define is a generalization of the consistency condition as defined for a substitution (see Definition 4.2.12). For simple updates, i.e. function and remove updates, it is fairly simple to define such consistency conditions. For extension updates, however, it is less obvious how to formulate these conditions.

We will follow the following strategy. First, we will give some examples of update sets, some of which are consistent, and some of which are not. This will give the reader some feeling of what exactly we are aiming at. Secondly, we
will formulate a semantic consistency condition which is very intuitive. Then, we will show how to simplify this condition. The resulting condition will serve as the definition of consistency. Finally, we will prove that the latter condition is equivalent to the former, intuitive one.

5.2.10 EXAMPLE Consider the following extension update, for which we assume that some signature exists to make it well-formed:

\[
\begin{align*}
\text{new } x : u \text{ with } \\
& f(x) := a \\
& f(b) := c
\end{align*}
\]

Since \( x \) is chosen to be a new element, it can never be the case that \( x = b \) in the current state (i.e. interpretation). So, supposing that \( b \) is defined in the current state, we have to conclude that this update is consistent with respect to the current state. From this we learn that a consistency condition must take into account that new elements like \( x \) are different from the existing ones.

Now, consider the following update:

\[
\begin{align*}
\text{new } x : u \text{ with } \\
& f(f(x)) := a \\
& f(b) := c
\end{align*}
\]

Although this update looks very much like the preceding one, it cannot be consistent with respect to any state. The reason is that update \( f(f(x)) := a \) cannot be performed: since \( f(x) \) is undefined for the new element \( x \), it is impossible to modify \( f \) for the argument \( f(x) \). On the other hand, the following update is consistent:

\[
\begin{align*}
\text{new } x : u \text{ with } \\
& a := f(x) \\
& f(b) := c
\end{align*}
\]

Here, \( a \) gets assigned the 'value' undefined.

From the last two examples we learn that a consistency condition must somehow incorporate the fact that a new, just created element cannot be in the domain of any function.

Our next example involves a contraction update:

\[
\begin{align*}
\text{rem } a : u \\
& f(b) := a \\
& g(a) := c
\end{align*}
\]

This update set is inconsistent for two reasons. First, the element which is denoted by \( a \) is being removed, while at the same time \( f(b) \) is set to that element. Secondly, \( g \) is set to the value of \( c \) for the argument denoted by \( a \), which is being removed simultaneously.

Finally, we consider the case in which two elements are removed from the same universe:
rem \( a : u \)
rem \( b : u \)

Certainly there will be no problem if \( a \) and \( b \) denote different elements in the universe denoted by \( u \). The question is what happens if \( a \) and \( b \) denote the same element. In that case \( a \) and \( b \) are evaluated in parallel, and then deleted in parallel. So, these updates should be consistent. This is in agreement with the fact that identical function updates are also consistent.

5.2.11 Convention If new \( x : u \) with \( U \in \text{UPD} \), then \( x \) is called the extension variable.

We will now develop an intuitive notion of consistency, taking the examples above into account. The idea is as follows. Firstly, we reduce an update set \( U \) containing extension updates to an update set solely consisting of simple updates. This will be done by first renaming all extension variables to fresh ones — ensuring that no name clashes can occur later — , and then stripping away all occurrences of 'new \( x : u \) with'. The stripped version of \( U \) will be referred to as \( U^x \), where \( x \) denote the new, renamed extension variables.

As a second step we define a condition \( C^+(U) \) which is defined for sets \( U \) of simple updates. If this condition is true in an interpretation \( \mathcal{I} \), and \( U \) does not contain extension variables, then the updates \( U \) can be executed on \( \mathcal{I} \). This condition is a generalization of the consistency condition for substitutions (see Definition 4.2.12).

If \( U \) does contain extension variables, say \( x \), we have to ensure that these variables denote 'new' elements. This is accomplished by extending the interpretation \( \mathcal{I} \) with new elements which we will use as denotations of the extension variables \( x \). Following this procedure, we say that the updates \( U \) are consistent with respect to an interpretation \( \mathcal{I} \), if the condition \( C^+(U^x) \) is true in the extended interpretation.

5.2.12 Definition Let \( U \in \text{UPD} \). A strip of \( U \) can be obtained by performing the following procedure:

1. Rename all extension variables in \( U \) such that they are mutually different and such that they are not occurring free in \( U \). Let \( \mathcal{V} \) be the resulting update set with renamed extension variables \( x \).

2. Strip all occurrences of the form 'new \( x : u \) with' from \( \mathcal{V} \).

The strip which results is referred to as \( U^x \).

Note that if \( U \) does not contain extension updates, any strip of \( U \) is equal to \( U \) itself. This fact is also reflected by our notation.

5.2.13 Definition Let \( U \subseteq \text{fin UPD} \) and let \( U \) have the following form:

\[
\{ c_i := r_i \mid c_i \in \text{VAR}_{u_i} \cup \text{CON}_{u_i} \} \\
\cup \{ f_j s_j := t_j \mid f_j \in \text{FUN}_{w_j, u_j} \land s_j \in \text{TERM}^{w_j} \land t_j \in \text{TERM}_{u_j} \} \\
\cup \{ \text{rem} q_k : u_k \mid q_k \in \text{TERM}_{w_k} \}
\]
where \( i, j \) and \( k \) range over some fixed, finite sets. Then the **naive consistency condition** for \( \mathcal{U} \), notation \( \mathcal{C}^+(\mathcal{U}) \), is defined by the following conjunction:

\[
\bigwedge_k q_k \land \bigwedge_j s_j \land \bigwedge_{c_{i_1} = c_{i_2}, i_1 \neq i_2} r_{i_1} \approx r_{i_2} \land \bigwedge_{f_{j_1} = f_{j_2}, j_1 \neq j_2} (s_{j_1} = s_{j_2} \rightarrow t_{j_1} \approx t_{j_2}) \\
\land \bigwedge_{i, k} r_{i} \neq q_k \land \bigwedge_{j, k, m} s_{j, m} \neq q_k \land \bigwedge_{j, k} t_{j} \neq q_k,
\]

where the conjuncts in the second line have to be understood as only containing well-formed inequalities (i.e. \( s \neq t \) only appears in a conjunct if \( s, t \in \text{TERM}_u \) for some \( u \in \text{SORT} \)), and where \( s_j = (s_{j_1}, \ldots, s_{j_m}, \ldots, s_{j_n}) \).

It is not hard to see that \( \mathcal{C}^+(\mathcal{U}) \) indeed is a generalization of the consistency condition for substitutions. For, in case that \( \mathcal{U} \) only contains function updates, there is a strong similarity between both definitions. In fact, a substitution can be seen as a restricted set of function updates. The difference is that in substitutions multiple occurrences of left hand side terms are forbidden, while in EA updates these are allowed. For this reason, the condition \( \mathcal{C}^+(\mathcal{U}) \) also contains the conjunct:

\[
\bigwedge_{c_{i_1} = c_{i_2}, i_1 \neq i_2} r_{i_1} \approx r_{i_2}.
\]

**5.2.14 Convention** Let \( \mathcal{I} = \langle \mathcal{A}, \beta \rangle \) be a \( \Sigma \)-interpretation, and let \( u = (u_1, \ldots, u_n), \ d = (d_1, \ldots, d_n) \), and \( x = (x_1, \ldots, x_n) \) be such that \( d_i \neq d_j \) when \( i \neq j \), \( d_i \notin \mathcal{A}_{u_i}, \ x_i \in \text{VAR}_{u_i} \), and \( x_i \neq x_j \) when \( i \neq j \). Then we will write

\[
\mathcal{I}^u_E[d] \rightarrow d
\]

as an abbreviation of

\[
\mathcal{I}^u_{d_1}[x_1 \rightarrow d_1] \ldots \mathcal{I}^u_{d_n}[x_n \rightarrow d_n].
\]

Note, that the conditions in this convention are such that the extension operators are defined, and that the order of appearance of them is immaterial.

Continuing our discussion of consistency and using the notations introduced above, we now could define consistency as follows.

Let \( \mathcal{I} \) be a \( \Sigma \)-interpretation, and let \( \mathcal{U} \subseteq \text{fn} \ \text{UPD} \). Furthermore, let the elements of \( x \) be the renamed extension variables of \( \mathcal{U} \). Then \( \mathcal{U} \) is **naively consistent** with respect to \( \mathcal{I} \), if for some \( d \) obeying the conditions of Convention 5.2.14:

\[
\mathcal{I}^u_E[x \rightarrow d] \models \mathcal{C}^+(\mathcal{U}^x).
\]

Using Theorem 4.2.17 it is easy to show that this naive definition of consistency is not dependent on the choice of the renamed extension variables. Later, we
will see that the actual choice of the elements \( d \) is immaterial as well (see Theorem 5.2.20).

Although we could use this definition for defining consistency, we will not do that. The reason for this will be explained in the following example.

**5.2.15 Example** We will demonstrate our semantic notion of consistency by the following sample update set \( \mathcal{U} \) (in which only one sort \( u \) is assumed):

\[
\text{new } x : u \text{ with } \\
f(x) := a \\
\text{rem } b : u \\
\text{new } x : u \text{ with } \\
f(x) := c \\
f(x) := a
\]

Renaming the extension variables results in:

\[
\text{new } y : u \text{ with } \\
f(y) := a \\
\text{rem } b : u \\
\text{new } z : u \text{ with } \\
f(z) := c \\
f(x) := a
\]

The strip \( \mathcal{U}^{(y,z)} \) of this update set is:

\[
f(y) := a \\
\text{rem } b : u \\
f(z) := c \\
f(x) := a
\]

The naive consistency condition \( C^+(\mathcal{U}^{(y,z)}) \) now is:

\[
\downarrow x \land \downarrow y \land \downarrow z \land \downarrow b \land (y = z \rightarrow a \approx c) \land (x = y \rightarrow a \approx a) \\
\land (x = z \rightarrow a \approx c) \land b \neq x \land b \neq y \land b \neq z \land b \neq a \land b \neq c.
\]

According to our tentative definition \( \mathcal{U} \) is naively consistent with respect to \( \mathcal{I} \) if the following holds:

\[
\mathcal{I} E^u_d[y\rightarrow d] E^u_e[z\rightarrow e] \models C^+(\mathcal{U}^{(y,z)}).
\]

Now, this statement contains some redundant information, as

\[
\mathcal{I} E^u_d[y\rightarrow d] E^u_e[z\rightarrow e] \models \downarrow y \land \downarrow z \land (y = z \rightarrow a \approx c) \land (x = y \rightarrow a \approx a) \\
\land (x = z \rightarrow a \approx c) \land b \neq y \land b \neq z
\]

is true by virtue only of the definition of \( \mathcal{I} E^u_d[y\rightarrow d] E^u_e[z\rightarrow e] \).

Using this observation, performing simple logical operations, and applying Corollary 4.2.24, we can simplify the statement to:

\[
\mathcal{I} \models \downarrow x \land \downarrow b \land b \neq x \land b \neq a \land b \neq c.
\]
This statement does not contain any extension variables. Moreover, it is considerably simpler that the original one. We will use this idea as basis for our definition of consistency.

5.2.16 Definition Let \( \mathcal{U} \subseteq_{\text{fin}} \text{UPD} \). Furthermore, let \( \mathcal{U}^x \) be a strip of \( \mathcal{U} \), and let \( \mathbf{x} = (x_1, \ldots, x_n) \). Then the consistency condition \( C(\mathcal{U}) \) for \( \mathcal{U} \) is obtained by applying to \( C^+(\mathcal{U}^x) \) the following procedure, in which \( x_i \) is supposed to be any variable in \( \mathbf{x} \):

1. Replace all formulae of the form \( \downarrow x_i \) by \( \text{tt} \).
2. Replace all formulae of the form \( \downarrow t \), where \( x_i \in \text{FV}(t) \) and \( t \neq x_i \), by \( \text{ff} \).
3. Replace all formulae of the form \( x_i = x_i \) by \( \text{tt} \).
4. Replace all formulae of the form \( s = t \), such that \( x_i \in \text{FV}(s) \cup \text{FV}(t) \), and such that \( s \equiv t \equiv x_i \), by \( \text{ff} \).

5.2.17 Remark It is obvious that after performing the procedure above upon \( C^+(\mathcal{U}^x) \) the result \( C(\mathcal{U}) \) does not contain extension variables anymore. That is also why we write \( C(\mathcal{U}) \), rather than \( C(\mathcal{U}^x) \).

Furthermore, note that in performing the procedure, expressions of the form \( s \neq t \) or \( s \approx t \) are abbreviations of, respectively, \( \neg(s = t) \) and \( \downarrow s \lor \downarrow t \rightarrow s = t \).

Finally, note that it is possible to simplify the result of performing the procedure by applying all possible obvious logical operations to it. For example, implications of the form \( \text{ff} \rightarrow \varphi \) or \( \varphi \rightarrow \text{tt} \) can be simplified to \( \text{tt} \).

5.2.18 Example Continuing with Example 5.2.15, and performing the four steps of Definition 5.2.16, the consistency condition \( C(\mathcal{U}) \) becomes:

\[
\downarrow x \land \text{tt} \land \text{tt} \land \downarrow b \land (\text{ff} \rightarrow a \approx c) \land (\text{ff} \rightarrow a \approx a) \\
\land (\text{ff} \rightarrow a \approx c) \land b \neq x \land \text{tt} \land \text{tt} \land b \neq a \land b \neq c.
\]

This can be simplified to:

\[
\downarrow x \land \downarrow b \land b \neq x \land b \neq a \land b \neq c,
\]

which is, not surprisingly, the same result as we found in Example 5.2.15.

5.2.19 Definition Let \( \mathcal{I} \) be a \( \Sigma \)-interpretation, and let \( \mathcal{U} \subseteq_{\text{fin}} \text{UPD} \). Then \( \mathcal{U} \) is consistent with respect to \( \mathcal{I} \), notation \( \text{Cons}(\mathcal{U}, \mathcal{I}) \), if

\[
\mathcal{I} \models C(\mathcal{U}).
\]

The next theorem states that the notion of consistency is equivalent to the notion of naive consistency. This theorem will enable us to use the latter notion when desirable.
5.2.20 **Theorem** Let \( \mathcal{I} \) be a \( \Sigma \)-interpretation, and let \( \mathcal{U} \subseteq_{\text{fin}} \mathcal{U} \mathcal{D} \). Then for any \( d \) obeying the conditions of Convention 5.2.14:

\[
\text{Cons} (\mathcal{U}, \mathcal{I}) \iff \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models \mathcal{C}^+ (\mathcal{U}^x).
\]

**Proof** By the definition of consistency just given above, \( \text{Cons} (\mathcal{U}, \mathcal{I}) \) means that \( \mathcal{I} \models \mathcal{C} (\mathcal{U}) \), where \( \mathcal{C} (\mathcal{U}) \) is the consistency condition for \( \mathcal{U} \). It directly follows from the procedure described in Definition 5.2.16, that there has to be a one to one correspondence between the atomic formulae of \( \mathcal{C} (\mathcal{U}) \) and those of \( \mathcal{C}^+ (\mathcal{U}^x) \). We will show that all atomic formulae in \( \mathcal{C}^+ (\mathcal{U}^x) \) have the same truth values in \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \) as the corresponding ones in \( \mathcal{C} (\mathcal{U}) \) have in \( \mathcal{I} \). From this fact the theorem follows.

For an atomic formula \( \varphi \) occurring in \( \mathcal{C}^+ (\mathcal{U}^x) \) we will consider two possibilities: \( \varphi \) does not contain an element \( x_i \) of \( x \), or \( \varphi \) does contain such a \( x_i \). In the first case, we have by Corollary 4.2.24:

\[
\mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models \varphi \iff \mathcal{I} \models \varphi.
\]

If, on the other hand, a variable \( x_i \) of \( x \) occurs in \( \varphi \), we have the following possibilities for \( \varphi \):

- \( \varphi \equiv \downarrow x_i \). By construction of \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \), we have in this case that \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models \downarrow x_i \). By Definition 5.2.16, the formula in \( \mathcal{C} (\mathcal{U}) \) corresponding to \( \downarrow x_i \) is tt. Obviously, we have \( \mathcal{I} \models \text{tt} \).

- \( \varphi \equiv \downarrow t \), such that \( x_i \in \text{Fv} (t) \) but \( t \neq x_i \). Here the corresponding formula in \( \mathcal{C} (\mathcal{U}) \) is ff. Now, we have neither \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models \downarrow t \), nor \( \mathcal{I} \models \text{ff} \).

- \( \varphi \equiv (x_i = x_i) \). The corresponding formula in \( \mathcal{C} (\mathcal{U}) \) is tt. Analogously as in first case, we have \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models x_i = x_i \) and \( \mathcal{I} \models \text{tt} \).

- \( \varphi \equiv (s = t) \), such that \( x_i \in \text{Fv} (s) \cup \text{Fv} (t) \), and such that not \( s \equiv t \equiv x_i \). Now, the corresponding formula is ff. Like in the second case above, we have neither \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \models s = t \), nor \( \mathcal{I} \models \text{ff} \).

We conclude that all atomic formulae in \( \mathcal{C}^+ (\mathcal{U}^x) \) have the same truth values in \( \mathcal{I} \mathcal{E}_d^w [x \mapsto d] \) as the corresponding ones in \( \mathcal{C} (\mathcal{U}) \) have in \( \mathcal{I} \). This means that we are done.

Any formula \( \varphi \in \text{FORM} \) determines a class of interpretations in which \( \varphi \) is true. So, it is sensible to define when a set of updates is consistent with respect to a formula.

5.2.21 **Definition** Let \( \varphi \in \text{FORM} \), and let \( \mathcal{U} \subseteq_{\text{fin}} \mathcal{U} \mathcal{D} \). Then \( \mathcal{U} \) is called **consistent** with respect to \( \varphi \), notation \( \text{Cons} (\mathcal{U}, \varphi) \), if

\[
\varphi \models \mathcal{C} (\mathcal{U}).
\]

The following proposition states that the notion of consistency is adequate in the sense that subsets of consistent sets of updates are also consistent.
5.2.22 **Proposition** Let $\mathcal{U}, \mathcal{V} \subseteq \text{fin UPD}$ and $\mathcal{U} \subseteq \mathcal{V}$. Moreover, let $\mathcal{I}$ be a $\Sigma$-interpretation. Then:

$$\text{Cons} (\mathcal{V}, \mathcal{I}) \Rightarrow \text{Cons} (\mathcal{U}, \mathcal{I}).$$

**Proof** Let $C(\mathcal{U})$ and $C(\mathcal{V})$ be the consistency conditions for $\mathcal{U}$ and $\mathcal{V}$, respectively. Then, by construction, we have that all conjuncts in $C(\mathcal{U})$ also appear in $C(\mathcal{V})$. This implies that $C(\mathcal{V}) \models C(\mathcal{U})$. From this fact, the proposition directly follows. □

5.2.23 **Corollary** Let $\mathcal{U}, \mathcal{V} \subseteq \text{fin UPD}$ and $\mathcal{U} \subseteq \mathcal{V}$. Moreover, let $\varphi \in \text{FORM}$. Then:

$$\text{Cons} (\mathcal{V}, \varphi) \Rightarrow \text{Cons} (\mathcal{U}, \varphi).$$

**Proof** This corollary follows from Proposition 5.2.22. □

5.2.24 **Convention** Let $R \in \text{RULE}$ be the update if $\varphi$ then $\mathcal{U}$, then (following [GR93]) we will write:

$$R? = \varphi,$$

$$R! = \mathcal{U}.$$

5.2.25 **Definition** A rule $R \in \text{RULE}$ is called **consistent**, if $\text{Cons} (R!, R?)$.

5.2.26 **Definition** An evolving algebra is called **consistent** if all its rules are consistent.

Any EA can be made consistent in a trivial manner by replacing the guards of all rules by $\text{ff}$. Of course this is not an interesting way of making them consistent. A non trivial manner is displayed in the next proposition.

5.2.27 **Proposition** Let $\mathcal{R}$ be an EA. Then the EA which is obtained by replacing every rule $R \in \mathcal{R}$ by:

$$\text{if } R? \land C(R!) \text{ then } R!.$$ 

is consistent.

5.2.28 **Example** The first rule of the EA $\mathcal{R}$ of Example 5.1.18 clearly is consistent. The second one is not consistent for we do not have:

$$\text{input} = \text{nil} \land \text{top} \neq \text{bottom} \models \downarrow \text{top} \land \text{top} \neq \text{next}(\text{top}).$$

We could make the rule consistent by adding $\downarrow \text{top} \land \text{top} \neq \text{next}(\text{top})$ to the guard.
The EA from the example above shows that strong consistency is perhaps not the notion of consistency for rules and EAs we are looking for. The question is how should consistency be used in our theory. Although this problem will be covered in the next chapter where a proof system for EAs will be introduced, we will already make some remarks about it.

An EA is a set of rules the purpose of which is that they are going to be executed repeatedly until no rule is applicable anymore. This idea makes only sense if there is an initial state where the execution starts. It is obvious that not all initial states will produce sensible computations. For instance, our EA of Example 5.1.18 should be applied to initial states for which the universe Stack contains an element which is both labeled top and bottom. Moreover, the constant input should be defined, and the constant output should be the empty string in initial states. These constraints could be coded in a formula called Precondition.

This suggests that all rules of an EA should be consistent with respect to their guards and the precondition, in other words:

\[ \text{Cons} (R!, R? \land \text{Precondition}) \]

But this requirement is still too restrictive, for Precondition describes the class of admissible initial states, and says nothing about intermediate states. So, what we actually want is an invariant, which in fact describes the class of states in which the execution of the EA takes place. For the invariant the following must hold:

\[ \text{Precondition} \models \text{Invariant} \]

Now, consistency of a rule will be related to the invariant as follows:

\[ \text{Cons} (R!, R? \land \text{Invariant}) \]

Of course, we want that in the state which is the result of applying all updates in \( R! \), the invariant still holds. But that topic has to wait until the next chapter. The point of this discussion is that the notion of consistency we are looking for is a relative one. This will be made precise in the following definition.

5.2.29 DEFINITION A rule \( R \in \text{Rule} \) is called consistent relative to the formula \( \varphi \in \text{FORM} \), if \( \text{Cons} (R!, R? \land \varphi) \).

5.2.30 DEFINITION An evolving algebra is called consistent relative to the formula \( \varphi \in \text{FORM} \), if all its rules are consistent relative to \( \varphi \).

5.2.31 DEFINITION An evolving algebra \( \mathcal{R} = \{R_1, \ldots, R_n\} \) is called deterministic if for all \( R_i, R_j \in \mathcal{R} \) such that \( i \neq j \):

\[ R_i? \models \neg R_j?. \]
We will now turn to our main task of this section: describing the operational semantics of EAs. In order to get an idea of what it is that we have to formalize, we remind the reader that the operation of an EA consists of a sequence of rule applications to states. This implies that the semantics must have a serial character. On the other hand an application of a rule consists of a parallel execution of updates. So, our semantics must have parallel features as well. How to combine these two different features is the problem we want to solve.

The approach we propose is to separate the parallel features from the serial ones. This is done by a two-layered semantics: the first layer will be used to describe the effect of the parallel execution of the updates of a rule, and the second layer will be a description of the effect of serially applying the rules of an EA. The kind of formalism in which we will express the semantics, will be Structural Operational Semantics. This kind of semantics was pioneered by Plotkin (see for example [Plo83]), and is covered nowadays in many textbooks on the semantics of programming languages (see, for example, [Hen90, NN92, SK95] for introductions, and [Ast91] for a more comprehensive account).

In our version of EAs, extension updates are, in fact, compound updates, i.e. they are structured. It is possible that this kind of updates has to be executed in parallel with other updates, possibly structured as well. This poses the problem of how to describe the effect of a parallel execution of actions which can be arbitrarily complex. To overcome this problem, we will describe the parallel effect in a serial way, such that the structure of the updates is respected. This means, for instance, that the result of an extension update will be dependent upon a subcomputation which computes the effect of the parallel execution of the updates nested in this extension update. Since, in our semantics, updates are being executed serially and in arbitrary order, we have, of course, to be sure that the order of executing these updates is immaterial with respect to the result.

To be more concrete, the effect of applying an update of a rule to a state, i.e. a $\Sigma$-interpretation, will be expressed by the relation:

$$\xrightarrow{\mathcal{I}}$$

where $\mathcal{I}$ denotes the state with respect to which the update will be evaluated. This can be explained as follows. Let $R \equiv \varphi$ then $U$ be a rule of some EA $\mathcal{R}$ and let $U = \{U_1, \ldots, U_n\}$ be the set of updates of $R$, then the effect of $R$ is described by a series:

$$\langle U_1, \ldots, U_n | \mathcal{I}_0 \rangle \xrightarrow{\mathcal{I}_0} \langle U_2, \ldots, U_n | \mathcal{I}_1 \rangle$$

$$\vdots$$

$$\xrightarrow{\mathcal{I}_0} \langle U_n | \mathcal{I}_{n-1} \rangle$$

$$\xrightarrow{\mathcal{I}_0} \langle \emptyset | \mathcal{I}_n \rangle,$$

where each state $\mathcal{I}_i$ is the result of applying update $U_i$ upon state $\mathcal{I}_{i-1}$ for $1 \leq i \leq n$. Of course, the guard $\varphi$ of $R$ has to be true in $\mathcal{I}_0$, and the updates
$U$ have to be consistent with respect to $I_0$. In this way, the effect of a rule is reduced to the effects of the individual updates of that rule. Note that the superscript $I_0$ of each arrow in the series is necessary. Since we want to describe parallelism, every update $U_i \in U$ has to be evaluated in the original state $I_0$ on which $U$ is to be executed in parallel.

Using the reflexive and transitive closure $\rightarrow^*_{I_0}$ of the relation $\rightarrow_{I_0}$, the transition from $I_0$ to $I_n$ effectuated by $R$ can be written as:

$$\langle U | I_0 \rangle \rightarrow^*_{I_0} \langle \emptyset | I_n \rangle.$$

The effect of local function updates and contraction updates can directly be expressed by local modification operators or assignment updates, and remove operators, respectively. For extension updates the situation is more complicated.

Suppose that the extension update new $x : u$ with $U$ has to be executed upon state $I$. Now, the idea is that $I$ has to be extended with a new element $d$ which will be temporarily bound to $x$. This yields the slightly different interpretation $I\mathcal{E}_d^u[x \rightarrow d]$. Then the updates of $U$ can be evaluated with respect to this new interpretation and executed upon it. This makes sense, for $U$ will probably contain occurrences of $x$, and these, as we just have seen, will be interpreted as the new value $d$. If all updates of $U$ have been executed, then $x$ will be reset to its original value in $I$, if that value still exists after performing these updates; otherwise, $x$ will be set to the ‘undefined value’ (note that $x$ will also be set to this ‘undefined value’, if $x$ was already undefined in $I$). The resulting interpretation is the effect of executing new $x : u$ with $U$ upon $I$.

With respect to the above mentioned ‘new element’ $d$, we will not only assume that $d$ is not a member of the actual state $I$, but also that $d$ has not already been an element of an earlier state. This assumption ensures that new elements cannot mistakenly be held for old elements that have been removed earlier. This is of importance with respect to the semantics of extension updates: the variable $x$ has to be reset to its old value if it still exists. In case the old element had been removed and could be taken back as a new element later, one would not be able to decide whether the old value still existed by only looking at the state resulting from the updates in the extension update.

This idea is formalized by attaching a set of available reserve elements to each state of the computation concerned. For the initial state of a computation a countably infinite set of reserve elements is fixed. This set has to be disjoint from all universes of that initial state. At each invocation of an extend operator an element of this set is taken out of it. Since reserve elements can only be taken out of the set of available reserve elements, the set of reserve elements attached to a state is always disjoint from the universes of that state. Moreover, elements discarded by a remove operator can never become reserve elements. Using this idea, transitions will look like as follows:

$$\langle U | I_0, D_0 \rangle \rightarrow^*_{I_0} \langle \emptyset | I_n, D_n \rangle,$$
where \( D_0, \ldots, D_n \) are the sets of available reserve elements attached to the states \( I_0, \ldots, I_n \), respectively.

The second level of our semantics describes the process of executing an evolving algebra \( R \). If \( R \) is a rule of the EA \( R, I \) and \( J \) are states, and \( D \) and \( D' \) are sets of reserve elements, then we will write:

\[
(I, D) \xrightarrow{R} (J, D'),
\]

if rule \( R \) transforms state \( I \) into \( J \). But this is expressible by the first level transition:

\[
\langle R! \mid I, D \rangle \xrightarrow{I} \langle \emptyset \mid J, D' \rangle.
\]

As we have noted before, \( R \) can only work upon state \( I \) if the guard \( R' \) of \( R \) is true in state \( I \), and if the updates \( R! \) of the rule are consistent with respect to \( I \). We will prove that consistency is a sufficient condition for the effect of a rule being well-defined, i.e. that the rule can be executed at all, and that the application order of the individual updates of the rule is immaterial.

Again we can consider the reflexive and transitive closure \( \xrightarrow{R^*} \) of the relation \( \xrightarrow{R} \). Then:

\[
(I, D) \xrightarrow{R^*} (J, D')
\]

holds if there exist states \( I_0, I_1, \ldots, I_n \) \((n \geq 0)\) such that:

\[
(I, D) = (I_0, D_0) \xrightarrow{R} (I_1, D_1) \xrightarrow{R} \ldots \xrightarrow{R} (I_n, D_n) = (J, D').
\]

Provided that there does not exist a state \( K \) such that \((J, D') \xrightarrow{R} (K, D'')\), we call the sequence \( I_0, I_1, \ldots, I_n \) a run or computation of \( R \). Moreover, \( I_0 \) is called the initial state, and \( I_n \) is called the final state of the run. Note that the initial set \( D_0 \) of reserve elements has to be disjoint from all universes of \( I_0 \).

Comparing the two relations \( \xrightarrow{I} \) and \( \xrightarrow{R} \), we may conclude that the first one gives us a fine-grained view on the effect of a rule, and that the second one gives us a more coarse-grained view on the effect of an EA.

5.2.32 Convention If \( U \in \text{UPD} \) and \( U \subseteq \text{fin UPD} \), then the expression \( U, U \) is used to denote \( \{U\} \cup U \) if \( U \notin U \).

5.2.33 Definition Let a signature \( \Sigma \) be given. Furthermore, let \( I, J \) and \( K \) be standard \( \Sigma \)-interpretations, or, for short, states, let \( U, V \subseteq \text{fin UPD} \), and let \( D \) denote a set, the elements of which are called reserve elements. Then the operational semantics of evolving algebras over \( \Sigma \) is defined by the following relations and rules. Rules of the form:

\[
P\quad C
\]

have to be understood as being conditional: if \( P \) is satisfied, then \( C \) holds. Put in another way: \( P \) is the premiss of the rule, and \( C \) the conclusion. Rules without a premiss could be considered axioms.
The relation \( \xrightarrow{t} \)

For every kind of update there is exactly one rule:

Rule for local function updates

\[
\langle s := t, \mathcal{U} | \mathcal{J}, D \rangle \xrightarrow{t} \langle \mathcal{U} | \mathcal{J}[s := t]^t, D \rangle.
\]

Rule for contraction updates

\[
\langle \text{rem } t : u, \mathcal{U} | \mathcal{J}, D \rangle \xrightarrow{t} \begin{cases} 
\langle \mathcal{U} | \mathcal{J} \mathcal{D}_t^u, D \rangle & \text{if } t^t \in |\mathcal{J}|_u, \\
\langle \mathcal{U} | \mathcal{J}, D \rangle & \text{otherwise.}
\end{cases}
\]

Rule for extension updates

\[
\langle \mathcal{U} | \mathcal{J} \mathcal{E}_d^u[x \mapsto d], D \setminus \{d\} \rangle \xrightarrow{\mathcal{J} \mathcal{E}_d^u[x \mapsto d]} \langle \emptyset | \mathcal{K}, D' \rangle \\
\langle \text{new } x : u \text{ with } \mathcal{U}, \mathcal{V} | \mathcal{J}, D \rangle \xrightarrow{t} \langle \mathcal{V} | \mathcal{K}[x \mapsto e], D' \rangle,
\]

where \( d \in D \) and:

\[
e = \begin{cases} 
x^t & \text{if } x^t \in |\mathcal{K}|_u, \\
\infty & \text{otherwise.}
\end{cases}
\]

In the last rule, \( \xrightarrow{t} \) denotes the reflexive and transitive closure of the relation \( \xrightarrow{t} \).

Note that it is supposed that a rule is applicable if and only if all operations which are applied to an interpretation mentioned in that rule, are defined, and if the set \( D \) of reserve elements is disjoint from all universes of \( \mathcal{J} \).

The relation \( \xrightarrow{R} \)

The only rule is:

\[
\langle R! | \mathcal{I}, D \rangle \xrightarrow{t} \langle \emptyset | \mathcal{J}, D' \rangle \\
\langle \mathcal{I}, D \rangle \xrightarrow{R} \langle \mathcal{J}, D' \rangle,
\]

provided that \( R \in R, \mathcal{I} \models R! \) and Cons \((R!, \mathcal{I})\).

\( \mathcal{J} \) is called a final state with respect to \( R \), if there is no interpretation \( \mathcal{K} \) such that:

\[
\langle \mathcal{J}, D \rangle \xrightarrow{R} \langle \mathcal{K}, D' \rangle.
\]

A computation or a run of \( R \) on interpretation \( \mathcal{I} \) is a series:

\[
\langle \mathcal{I}, D \rangle = (I_0, D_0) \xrightarrow{R} (I_1, D_1) \xrightarrow{R} \ldots \xrightarrow{R} (I_n, D_n) = (\mathcal{J}, D')
\]
where the set of initial reserve elements \( D \) is disjoint from all universes of \( I \), and where \( J \) is a final state and \( n \geq 0 \). Using the reflexive and transitive closure \( \xrightarrow{R} \) of the relation \( \xrightarrow{R} \), this series can be written as:

\[(I, D) \xrightarrow{R} (J, D').\]

State \( I \) is called the initial state of the run.

5.2.34 Remark As opposed to ‘semistandard’ interpretations which could also be defined for EAs, (cf. Section 4.4), standard interpretations are the most natural candidates for being called ‘states’. In actual implementations of evolving algebras only the dynamic universes, functions and constants will be implemented, however. The static functions and constants will be explicitly presented as values and routines, respectively (see e.g. [Vis96]).

5.2.35 Remark The transition rule for extension updates is non-deterministic: the actual choice of the element \( d \in D \) is free.

Note that the semantics of \( \text{rem } t : u \) covers the possibility of parallel contraction updates of the same element — recall from Example 5.2.10 that these parallel contractions are allowed. Since our semantics is sequential, a problem would occur if contraction updates would always be defined in terms of applications of the \( \text{D} \) operator, for if an element has already been removed, it cannot be removed a second time. This situation has been taken care of by stipulating that \( \text{rem } t : u \) has no effect if the object to be removed is not an element of the universe concerned, i.e. if there is nothing to delete.

With respect to the semantics of the extension update new \( x : u \) with \( U \), we already discussed the behaviour of the extension variable \( x \). The idea was that after performing the updates \( U \), the variable \( x \) would be reset to its original value. The problem, however, is that this original value is not guaranteed to exist after the execution of \( U \). The reason for this is, that \( U \) could contain a contraction update with the effect of precisely removing that original value. In such cases \( x \) is set to \( \infty \). Note that \( x \) is also set to \( \infty \) if its old value was \( \infty \).

The reader might have realized by now, that contraction updates are ‘difficult’. This is already apparent from the definition of the remove operator which describes the semantic effect of this kind of update. Obviously, without contraction updates life would be a lot easier.

Inspection of the definition of final state reveals that there are two possibilities for reaching a final state. The first possibility occurs if for all rules \( R \in \mathcal{R} \) we have \( J \models \neg R \). In this case we will say that \( \mathcal{R} \) terminates normally or that \( J \) is a normal final state.

In the second case there still is a rule \( R \in \mathcal{R} \) such that \( J \models R \), but not \( \text{Cons}(R!, J) \). Now, \( \mathcal{R} \) is said to terminate abnormally, and that \( J \) is an abnormal final state. Note, that for consistent EAs this situation cannot occur.

5.2.36 Definition Let \( \mathcal{R} \) be an EA, and let \((I, D) \xrightarrow{\mathcal{R}} (J, D')\) be a run of \( \mathcal{R} \) with final state \( J \). Then \( \mathcal{R} \) is said to terminate normally, and \( J \) is
said to be a normal final state, if \( \mathcal{J} \vdash R \) for all rules \( R \in \mathcal{R} \). In all other cases \( \mathcal{R} \) is said to terminate abnormally, and \( \mathcal{J} \) is said to be an abnormal final state.

**5.2.37 Convention** Let \( \mathcal{U}, \mathcal{V} \subseteq_{\text{fin}} \text{UPD} \) be sets of updates, let \( \mathcal{R} \) be an EA, let \( \mathcal{I}, \mathcal{J} \) and \( \mathcal{K} \) be \( \Sigma \)-interpretations, and let \( D \) and \( D' \) be sets of reserve elements. Then:

\[
\langle \mathcal{U} \mid \mathcal{J}, D \rangle \xrightarrow{\mathcal{I}} \langle \mathcal{V} \mid \mathcal{K}, D' \rangle \quad \text{and} \quad \langle \mathcal{J}, D \rangle \xrightarrow{\mathcal{R}} \langle \mathcal{K}, D' \rangle
\]

will be called transitions. Moreover, with respect to the first transition, structures like \( \langle \mathcal{U} \mid \mathcal{J}, D \rangle \) will be called configurations, and \( \mathcal{I} \) will be called the evaluation state of the transition.

**5.2.38 Definition** Let \( \mathcal{U} \subseteq_{\text{fin}} \text{UPD} \), and let \( \mathcal{I} \) and \( \mathcal{J} \) be \( \Sigma \)-interpretations. Moreover, let \( D_0 \) be the set of reserve elements in the initial state of some run, in which the transition:

\[
\ldots \xrightarrow{\mathcal{I}} \langle \mathcal{U} \mid \mathcal{J}, D_n \rangle \xrightarrow{\mathcal{I}} \ldots
\]

occurs. Then, an element \( d \in D_0 \) is called available with respect to \( \mathcal{J} \) and \( D_n \), if \( d \in D_n \).

**5.2.39 Convention** We will often speak of the availability of \( d \) with respect to a state \( \mathcal{J} \) without mentioning the actual set of reserve elements. In that case we will assume that a suitable set \( D_n \) of reserve elements exists.

**5.2.40 Remark** Definedness of \( E_d^u \) and availability of \( d \) are different notions. On the one hand, \( E_d^u \) is defined on \( \mathcal{J} \), if \( d \) is available with respect to \( \mathcal{J} \). The converse, on the other hand, does not have to hold: if \( d \) is not available with respect to \( \mathcal{J} \), then it is possible that \( d \not\in |\mathcal{J}|_u \) in which case we have that \( E_d^u \) is defined on \( \mathcal{J} \).

**5.2.41 Convention** To simplify notation, we will drop the actual sets of reserve elements in configurations, when possible. So, in fact we will use the following transition rules:

\[
\langle s := t, \mathcal{U} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \mathcal{U} \mid \mathcal{J}[s := t]^I \rangle,
\]

\[
\langle \text{rem } t : u, \mathcal{U} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \begin{cases} 
\langle \mathcal{U} \mid \mathcal{J}^D_t \rangle & \text{if } t^I \in |\mathcal{J}|_u, \\
\langle \mathcal{U} \mid \mathcal{J} \rangle & \text{otherwise},
\end{cases}
\]

\[
\langle \mathcal{U} \mid \mathcal{J} E_d^u[x \mapsto d] \rangle \xrightarrow{\mathcal{I}E_d^u[x \mapsto d]} \langle \emptyset \mid \mathcal{K} \rangle
\]

\[
\langle \text{new } x : u \text{ with } \mathcal{U}, \mathcal{V} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \mathcal{V} \mid \mathcal{K}[x \mapsto e] \rangle,
\]

\[
\langle R! \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle
\]

\[
\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{J}
\]
Of course, we assume for this rules the same requirements to be fulfilled as those in Definition 5.2.33. Moreover, we will assume that with respect to reserve elements the conventions discussed above will be followed. In particular, this means that in the rule for the extension update above the element $d$ has to be available with respect to state $\mathcal{J}$.

5.2.42 Example Recall the note below Definition 5.2.3 in which we remarked that variable assignments are partial. The reason for this can now be made clear. Consider the following extension update:

\[
\text{new } x : u \text{ with } x := \bot
\]

Although this update is not very interesting in itself — it leaves the current state almost unaltered by only adding a new, nameless element to one of its universes — its operational semantics is. We will apply the operational rules to show its effect. Since:

\[
\langle x := \bot | \mathcal{IE}_d^{u^p}[x \mapsto d] \rangle \xrightarrow{\mathcal{IE}_d^{u^p}[x \mapsto d]} \langle \emptyset | \mathcal{IE}_d^{u^p}[x \mapsto d][x \mapsto \infty] \rangle
\]

\[
\phantom{\langle x := \bot | \mathcal{IE}_d^{u^p}[x \mapsto d] \rangle \xrightarrow{\mathcal{IE}_d^{u^p}[x \mapsto d]} \langle \emptyset | \mathcal{IE}_d^{u^p}[x \mapsto \infty] \rangle} = \langle \emptyset | \mathcal{IE}_d^{u^p}[x \mapsto \infty] \rangle,
\]

we have:

\[
\langle \text{new } x : u \text{ with } x := \bot | \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset | \mathcal{IE}_d^{u^p}[x \mapsto \infty][x \mapsto x^I] \rangle
\]

\[
\phantom{\langle \text{new } x : u \text{ with } x := \bot | \mathcal{I} \rangle \xrightarrow{\mathcal{I}}} = \langle \emptyset | \mathcal{IE}_d^{u^p} \rangle.
\]

Clearly, in the intermediate state $\mathcal{IE}_d^{u^p}[x \mapsto \infty]$ the value of $x$ is undefined. Therefore, we need partial variable assignments. This motivates our choice of $E$-logic above $E^+\text{-logic}$ where free variables are always denoting. An other option would be to forbid updates of the kind displayed above. This would be an unacceptable ad hoc decision, in our opinion.

5.3 Adequacy results

The theorems that will be proven in this section, establish the adequacy of the operational semantics. The first of these, the Conversion Theorem, expresses that extension variables can be renamed without changing the semantics of the extension update considered. According to the Normal Form Theorem every set of updates has an operationally equivalent normal form. The Isomorphism Theorem says that transitions are invariant with respect to certain isomorphisms.

The Completeness Theorem states that the first layer of our semantics is complete in the following sense: if a set of updates is consistent with respect to an initial state, then the semantics is rich enough to provide a final state for this set of updates.

The Soundness Theorem, finally, says that the order of executing the updates of a rule is immaterial. In fact, this means that the effect of the parallel
execution of a set of updates is well-defined. Before and between these theorems and their proofs we will establish a number of useful lemmas.

The soundness and completeness results mentioned above concern the relation $\rightarrow^x$. At the end of this section we will also say something about these issues with respect to the relation $\rightarrow^r$.

Since almost all of our proofs are by induction on the complexity of a set of updates, we will first define this notion. Intuitively, the complexity of a set of updates is the maximum of all nesting depths of extension updates in that set.

**5.3.1 Definition** Let $\mathcal{U} \subseteq \text{fin} \ \text{UPD}$, then the complexity of $\mathcal{U}$, notation $\text{COMP} (\mathcal{U})$, is recursively defined by:

- $\text{COMP} (\{s := t\}) = 0$,
- $\text{COMP} (\{\text{rem } t : u\}) = 0$,
- $\text{COMP} (\{\text{new } x : u \text{ with } \mathcal{V}\}) = \text{COMP} (\mathcal{V}) + 1$,
- $\text{COMP} (\mathcal{U}, \mathcal{V}) = \max (\text{COMP} (\{\mathcal{U}\}), \text{COMP} (\mathcal{V}))$.

The first result of this section establishes the adequacy of the transition rules with respect to the sets of reserve elements.

**5.3.2 Proposition** Let $\mathcal{U}, \mathcal{V} \subseteq \text{fin} \ \text{UPD}$, let $\mathcal{I}, \mathcal{J}$ and $\mathcal{K}$ be $\Sigma$-interpretations, and let $D$ and $D'$ be sets of reserve elements, such that $D$ is disjoint from all universes of $\mathcal{J}$, and:

$$(\mathcal{U} \mid \mathcal{J}, D) \xrightarrow{\mathcal{I}} (\mathcal{V} \mid \mathcal{K}, D'),$$

then $D'$ is disjoint from all universes of $\mathcal{K}$ and $D' \subseteq D$.

**Proof** The proof is by induction on the complexity and the cardinality of the set of updates that is being executed, i.e. $\mathcal{U} \setminus \mathcal{V}$.

As a consequence of the proposition above, an element $d$ of the initial set of reserve elements (i.e. the set of reserve elements attached to the initial state of the run concerned) is not available with respect to $\mathcal{K}$ and $D'$, if $d \in D \setminus D'$ for some $\mathcal{U} \subseteq \text{fin} \ \text{UPD}$, $\Sigma$-interpretation $\mathcal{J}$, and set $D$ such that:

$$(\mathcal{U} \mid \mathcal{J}, D) \xrightarrow{\mathcal{I}} (\mathcal{V} \mid \mathcal{K}, D')$$

occurs in the transition concerned. This means that for members of the initial set of reserve elements the property of not being available can be identified with the property of being already used.

Another consequence of this proposition is that if, in the transition displayed above, $d$ is available with respect to $\mathcal{K}$ and $D'$, then $d$ is also available with respect to $\mathcal{J}$ and $D$.

We already explained that discarded elements will never become reserve elements, or put differently: once discarded, always discarded. This property is expressed by the following lemma.
5.3.3 Lemma Let $I$, $J_1$, $J_2$ and $J_3$ be $\Sigma$-interpretations such that $d \in |J_1|_u$ and $d \notin |J_2|_u$. Moreover, let $U, V, W \subseteq \operatorname{fin} U \cap D$ be such that:

$$
\left\langle U \mid J_1 \right\rangle \xrightarrow{I} \left\langle V \mid J_2 \right\rangle \xrightarrow{I} \left\langle W \mid J_3 \right\rangle.
$$

Then $d \notin |J_3|_u$.

Proof. We give an informal proof. A more rigorous proof could be obtained by induction using the transition rules given in Definition 5.2.33.

Let $D$ be the set of remaining reserved elements with respect to $J_1$, then clearly $d \notin D$, since $d \in |J_1|$. Now, since $d \notin J_2$, the object $d$ must have been discared by a contraction update in $U$. By Proposition 5.3.2, we have for the set $D'$ of remaining reserved elements with respect to $J_2$, that $D' \subseteq D$. This implies that $d \notin D'$, as $d \notin D$. The only possibility for $d$ to become an element of $J_3$ is by an extension update in $V$. This is, however, impossible since in that case we should have $d \in D'$.

We will now investigate the structure of end states of transitions. If we look at the transition rules of the operational semantics, we see that at any moment of execution of a local function update a corresponding local modification operator (or assignment update) is applied to the current state. Likewise, in case of a contraction update, a corresponding remove operator is applied (or no operator in case the element to be removed had already been removed).

With respect to extension updates the situation is more complicated. First, the current state is extended by a new element and the extension variable is set to the newly created element. Then, the updates in the scope of this extension update are executed. When this has been completed, the original value of the extension variable is restored (if this value still exists) or made undefined (if the original value has been removed during the execution). This procedure is applied recursively if the extension update contains other extension updates.

From this discussion it might be clear that the result of executing a set of updates to some initial state is equal to the result of applying a sequence of modification operators, remove operators, extension operators and assignment updates to that initial state. In the following example this will be illustrated.

5.3.4 Example Consider the following set $U$ of updates:

$$
f(a) := b
$$

new $x : u$ with

$$
b := x
$$

new $y : v$ with

$$
c := y
$$

rem $c : u$

Now, suppose that all these updates can be executed in the given order, and that:

$$
\left\langle U \mid I \right\rangle \xrightarrow{I} \left\langle \emptyset \mid J \right\rangle,
$$

then $J$ can be written as:
5.3. Adequacy results

\[ J = I \left[ f[a^{x \rightarrow b}] \right] \]
\[ E_0^u[x \rightarrow d'] \]
\[ [b \rightarrow x^{TE_0^e[x \rightarrow d]}] \]
\[ E_0^u[y \rightarrow e] \]
\[ [c \rightarrow y^{TE_0^e[x \rightarrow d]}E_0^u[y \rightarrow e]] \]
\[ [y \rightarrow e'] \]
\[ x \rightarrow d' \]
\[ D_{c,x}^u, \]

where \( d' \) and \( e' \) are in accordance with the rule for extension updates, and where it is supposed that \( c^x \) still exists before the application of the remove operator. Note how the nested structure of the updates is reflected in nested blocks of operators:

\[ E_0^u[x \rightarrow d] \ldots [x \rightarrow d'] \] and \[ E_0^u[y \rightarrow e] \ldots [y \rightarrow e']. \]

Every update is evaluated with respect to a state, the first one being \( I \) in this example. When a block is entered the evaluation state is extended and modified by an extend operator and an assignment update, respectively. Outside a block the evaluation state before entering the block is taken up again.

5.3.5 Lemma Let \( I \) and \( J \) be \( \Sigma \)-interpretations, and let \( U \subseteq_{\text{fin}} \text{UPD} \). Then:

\[ \langle U \mid I \rangle \xrightarrow{I} \langle \emptyset \mid J \rangle, \]

if and only if for some \( n \geq 0:\)

\[ J = IO_1 \ldots O_n, \]

where each \( O_i \) (\( 1 \leq i \leq n \)) is a local modification operator, an assignment update, an extend operator, or a remove operator, and where the sequence \( O_1 \ldots O_n \) is obtained by a sequence of applications of the transition rules of Definition 5.2.33 starting in configuration \( \langle U \mid I \rangle \) with evaluation state \( I \).

Proof The proof is by induction on the complexity \( \text{COMP} (U) \) of \( U \) using the transition rules.

Any operator \( O_i \) in the sequence \( O_1 \ldots O_n \) of the preceding lemma is inserted by an application of one of the transitions rules. In this way any local function update corresponds to an unique local modification operator or assignment update in the sequence. Conversely, any local modification operator corresponds to a local function update. Assignment updates, however, may correspond to local function updates of the form \( x := t \), but may also have been inserted by the transition rule for extension updates. Remove operators always correspond to contraction updates. The other way around does not hold, since some contraction updates do not result into an insertion of a remove operator.

This notion of ‘corresponding updates and operators’ can easily be formalized by labeling the operators in some suitable way. We will not do so, since the idea is simple and clear.
5.3.6 Lemma Let $I$ and $J$ be $\Sigma$-interpretations, and let $U \subseteq_{\text{fin}} \text{UPD}$. Moreover, let:

\[
\langle U \mid I \rangle \xrightarrow{J} \langle \emptyset \mid IO_1 \ldots O_n \rangle
\]

for some $n \geq 0$. Then the sequence $O_1 \ldots O_n$ of operators has the following properties:

1. If $\text{COMP}(U) = 0$, then any operator $O_i$ ($1 \leq i \leq n$) corresponds to a simple update (i.e. a local function update or a contraction update) in $U$.

2. If $O_i$ is the extend operator $E_d^n$, then $d \notin |IO_1 \ldots O_{i-1}|_u$.

3. If $O_i$ corresponds to a simple update $U \in U$, then $O_i$ is the result of interpreting $U$ with respect to a state $IE_d^n[x \mapsto d]$, such that $x$ are the extension variables belonging to the extension updates in which $U$ is nested.

4. If $O_i$ is the result of evaluating its corresponding, simple update $U$ with respect to the evaluation state $IE_d^n[x_1 \mapsto d_1] \ldots E_{d_n}^u[x_n \mapsto d_n]$, then there exist $3n$ operators:

\[
\begin{align*}
O_{k_1} &= E_{d_1}^u, \\
O_{k_1+1} &= [x_1 \mapsto d_1], \\
O_{m_1} &= [x_1 \mapsto d'_1], \\
\vdots & \hspace{1cm} \vdots \\
O_{k_n} &= E_{d_n}^u, \\
O_{k_n+1} &= [x_n \mapsto d_n], \\
O_{m_n} &= [x_n \mapsto d'_n]
\end{align*}
\]

such that $k_1 < k_1 + 1 < \ldots < k_n < k_n + 1 < i < m_1 < \ldots < m_n$, and where $d'_j$ ($1 \leq j \leq n$) denotes the value to which the extension variable $x_j$ is reset after completing the corresponding extension update.

Proof Again, the proof is by induction on the complexity $\text{COMP}(U)$ of $U$ using the transition rules.

If $U$ is an update set and $x$ is a variable that does not occur free in $U$, then one would expect that $x$ is not changed while executing $U$. This would be true if the value of $x$ would not be removed by a contraction update occurring in $U$.

5.3.7 Lemma Let $U \subseteq_{\text{fin}} \text{UPD}$, and let $x \in \text{VAR}_u$ such that $x \notin \text{FV}(U)$. Moreover, let $I$, $J$ and $K$ be $\Sigma$-interpretations such that:

\[
\langle U \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle.
\]

Then:

\[
x^K = \begin{cases} 
x^J & \text{if } x^J \in |K|_u, \\
\infty & \text{otherwise.}
\end{cases}
\]
5.3. Adequacy results

PROOF The proof is by induction over the complexity COMP (U) of U.

Base case: COMP (U) = 0

We prove the base case by induction on the number of updates #(U) in U. The base case is #(U) = 1 (if #(U) = 0, the lemma trivially holds). We have the following cases:

- U = {c := t}, where c is a individual constant or a variable. Since x \notin Fv (U), we infer that c \neq x. According to the operational semantics we have:
  \[
  \langle c := t \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \upharpoonright \mathcal{I} \rangle.
  \]
  It is clear from this transition that x^{\mathcal{J} \upharpoonright \mathcal{I}} = x^\mathcal{J}, which proves this case.

- U = \{ fs := t \}. Like the preceding case.

- U = \{rem t : v \}. According to our operational semantics we have:
  \[
  \langle \text{rem } t : v \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \begin{cases} 
  \langle \emptyset \mid \mathcal{J} D^v_{\mathcal{I}} \rangle & \text{if } t^\mathcal{I} \in |\mathcal{J}|_v, \\
  \langle \emptyset \mid \mathcal{J} \rangle & \text{otherwise.}
  \end{cases}
  \]
  This means we have two possibilities: t^\mathcal{I} \notin |\mathcal{J}|_v or t^\mathcal{I} \in |\mathcal{J}|_v. In the first case, the lemma clearly holds.
  In the second case we reason as follows. If v \neq u, we clearly have x^{\mathcal{J} D^v_{\mathcal{I}}} = x^\mathcal{J}. If v \equiv u, then the possibility exists that x^\mathcal{J} = t^\mathcal{I}, in which case x^{\mathcal{J} D^v_{\mathcal{I}}} = \infty. So, in this case, the lemma holds as well.

The induction step, #(U) = n + 1, is straightforward and left to the reader.

Induction step: COMP (U) = m + 1

Again, we use induction over #(U). The base case is #(U) = 1. This implies that U = \{new z : v with \mathcal{V}\}, with COMP (\mathcal{V}) = m. We will apply the induction hypothesis to \mathcal{V}. We consider two different cases for z:

- z \neq x. Suppose that the following transition is possible:
  \[
  \langle \text{new } z : v \text{ with } \mathcal{V} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{K} \rangle.
  \]
  Then, according to the operational semantics a transition of the following form must exist:
  \[
  \langle \mathcal{V} \mid \mathcal{J} E^v_d[z \leftrightarrow d] \rangle \xrightarrow{\mathcal{I} E^v_d[z \leftrightarrow d]} \langle \emptyset \mid \mathcal{K}' \rangle,
  \]
  such that \mathcal{K} = \mathcal{K}'[z := e], where e = z^\mathcal{J} if z^\mathcal{J} \in |\mathcal{K}'|_v, and e = \infty otherwise. Since x \notin Fv (U), we may apply the induction hypothesis to the last
transition yielding $x^{K'} = x^J E^*_d[x \mapsto d]$ if $x^J E^*_d[x \mapsto d] \in |K'|_u$ and $x^{K'} = \infty$ otherwise. On the other hand, as $z \not= x$, we also have that $x^K = x^{K'}$ and $x^J E^*_d[x \mapsto d] = x^J$. Moreover, it is clear that $|K'|_u = |K|_u$. Combining all these facts, we infer that $x^K = x^J$ if $x^J \in |K|_u$ and $x^K = \infty$ otherwise, which had to be proven.

- $z \equiv x$. Suppose that the following transition is possible:

$$\langle \text{new } x : u \text{ with } V \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle.$$

Then, according to the operational semantics we have $x^K = x^J$ if $x^J \in |K|_u$ and $x^K = \infty$ otherwise.

The induction step $\#(U) = n + 1$ is left to the reader. 

5.3.8 REMARK The lemma we just proved can easily be strengthened. In order to do so, we have to define the domain of an update set $U$. Informally, the domain of $U$ contains any individual constant or free variable which occurs as a left hand side of a local function update in $U$, or which occurs as the term in a contraction update in $U$. Inspection the proof of Lemma 5.3.7 reveals that the lemma holds for any individual constant or variable which does not occur in the domain of $U$.

The following is, in a way, a variant of the Lemma 5.3.7. It can be proven using the same technique. The proof is omitted.

5.3.9 LEMMA Let $U \subseteq \text{fin UPD}$, let $x \in \text{VAR}_u$ such that $x \not\in \text{FV}(U)$, and let $d \in |J|_u$ or $d = \infty$. Moreover, let $I$, $J$ and $K$ be $\Sigma$-interpretations such that:

$$\langle U \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle.$$

Then:

$$\langle U \mid J[x \mapsto d] \rangle \xrightarrow{I} \langle \emptyset \mid K[x \mapsto e] \rangle,$$

where:

$$e = \begin{cases} d & \text{if } d \in |K|_u, \\ \infty & \text{otherwise}. \end{cases}$$

The following lemma describes the effect of renaming a free variable in a set of updates into a fresh variable. This lemma is formulated in such way that in the proof which is by induction on the complexity of the update set, the induction step yields an induction hypothesis which is easy to use.

5.3.10 LEMMA Variable change

Let $U \subseteq \text{fin UPD}$, and let $x, y \in \text{VAR}_u$ such that $y \not\in \text{FV}(U)$. Moreover, let $I$, $J$ and $K$ be $\Sigma$-interpretations such that:

$$\langle U \mid J[x \mapsto d] \rangle \xrightarrow{I[x \mapsto e]} \langle \emptyset \mid K[x \mapsto d'] \rangle,$$
and:

\[ x^K = \begin{cases} 
  x^J & \text{if } x^J \in |K|_w, \\
  \infty & \text{otherwise.}
\end{cases} \]

Then:

\[ \langle U[x:=y] | J[y\mapsto d] \rangle \xrightarrow{I[y\mapsto e]} \langle \emptyset | K[y\mapsto d'] \rangle. \]

**Proof** The proof is by induction on the complexity \( \text{Comp} (U) \) of \( U \).

**Base case:** \( \text{Comp} (U) = 0 \)

We prove the base case by induction on the number of updates \( \#(U) \) in \( U \). The base case is \( \#(U) = 1 \) (if \( \#(U) = 0 \), the lemma trivially holds). We have the following cases:

- \( U = \{ x := t \} \). Here we have \( U[x:=y] = \{ y := t[x:=y] \} \). Suppose that the following transition is possible:

\[ \langle x := t \mid J[x\mapsto d] \rangle \xrightarrow{I[x\mapsto e]} \langle \emptyset \mid K[x\mapsto d'] \rangle, \]

where \( x^K = x^J \) if \( x^J \in |K|_w \) and \( x^K = \infty \) otherwise. According to our operational semantics, we should have:

\[ \langle x := t \mid J[x \mapsto d] \rangle \xrightarrow{I[x \mapsto e]} \langle \emptyset \mid J[x \mapsto d][x \mapsto t[I[x \mapsto e]]] \rangle \\
= \langle \emptyset \mid J[x \mapsto t[I[x \mapsto e]]] \rangle. \]

Combining this with transition 5.1, we see that \( d' = t[I[x \mapsto e]] \), and that \( K \) and \( J \) may only differ with respect to the denotation of \( x \). The latter property implies that all corresponding universes of \( K \) and \( J \) have to be equal, and therefore that \( x^J \in |K|_w \). From this we infer, using the requirement about \( x^K \), that \( x^K = x^J \). But this implies that \( K = J \).

On the other hand:

\[ \langle y := t[x:=y] \mid J[y\mapsto d] \rangle \xrightarrow{I[y\mapsto e]} \langle \emptyset \mid J[y\mapsto d][y\mapsto t[x:=y]I[y\mapsto e]] \rangle \\
= \langle \emptyset \mid J[y\mapsto t[I[y\mapsto e][x\mapsto yI[y\mapsto e]]]] \rangle \\
= \langle \emptyset \mid J[y\mapsto t[I[x\mapsto e]]] \rangle. \]

The last steps are justified by Theorem 4.2.17 and the fact that \( y \notin \text{Fv} (t) \).

Now, since \( K = J \) and \( d' = t[I[x\mapsto e]] \), we are done with this case.
• \( U = \{ fs := t \} \). Here we have \( U[x := y] = \{ fs[x := y] := t[x := y] \} \). Suppose that the following transition is possible:

\[
\begin{align*}
\langle fs := t \mid J[x \mapsto d] \rangle & \xrightarrow{I[x \mapsto e]} \langle \emptyset \mid K[x \mapsto d'] \rangle,
\end{align*}
\]

(5.2)
such that \( x^K \) satisfies the requirements. According to our operational semantics, we have:

\[
\begin{align*}
\langle fs := t \mid J[x \mapsto d] \rangle \\
\xrightarrow{I[x \mapsto e]} \langle \emptyset \mid J[x \mapsto d][fs[I[x \mapsto e] \mapsto t[I[x \mapsto e]]]] \rangle \\
= \langle \emptyset \mid J[f[s[I[x \mapsto e] \mapsto t[I[x \mapsto e]]]]] \rangle \langle x \mapsto d \rangle.
\end{align*}
\]

Combining this with transition 5.2, and using the same kind of argument as in the previous case, we infer that \( K = J[f[s[I[x \mapsto e] \mapsto t[I[x \mapsto e]]]]] \). Furthermore, we see that \( d' = d \).

On the other hand:

\[
\begin{align*}
\langle fs[x := y] := t[x := y] \mid J[y \mapsto d] \rangle \\
\xrightarrow{I[y \mapsto e]} \langle \emptyset \mid J[y \mapsto d][fs[x := y][I[y \mapsto e] \mapsto t[x := y][I[y \mapsto e]]]] \rangle \\
= \langle \emptyset \mid J[y \mapsto d][f[s[I[y \mapsto e][x \mapsto e] \mapsto t[I[y \mapsto e][x \mapsto e]]]]] \rangle \\
= \langle \emptyset \mid J[f[s[I[x \mapsto e] \mapsto t[I[x \mapsto e]]]]][y \mapsto d] \rangle.
\end{align*}
\]

This is what we had to show. Note, that we have used Theorem 4.2.17 and the fact that \( y \notin \text{FV}(t) \).

• \( U = \{ c := t \} \), where \( c \not\equiv x \): like the preceding case.

• \( U = \{ \text{rem } t : v \} \). In this case \( U[x := y] = \{ \text{rem } t[x := y] : v \} \). Suppose that:

\[
\begin{align*}
\langle \text{rem } t : v \mid J[x \mapsto d] \rangle & \xrightarrow{I[x \mapsto e]} \langle \emptyset \mid K[x \mapsto d'] \rangle,
\end{align*}
\]

(5.3)
such that \( x^K \) meets the requirements. Firstly, observe that by Theorem 4.2.17 and the fact that \( y \notin \text{FV}(t) \):

\[
\begin{align*}
t[x := y][I[y \mapsto e]] & = t[I[y \mapsto e][x \mapsto y][I[y \mapsto e]]] \\
& = t[I[x \mapsto e]]
\end{align*}
\]

This implies that \( t[I[x \mapsto e]] \in |J[x \mapsto d]|_v \) if and only if \( t[x := y][I[y \mapsto e]] \in |J[y \mapsto d]|_v \) (since \( J[x \mapsto d] \) and \( J[y \mapsto d] \) have identical universes).
Now suppose that \( t_{x \rightarrow e} \in \mathcal{J}[x \rightarrow d] \). Then, according to our operational semantics:

\[
\langle \text{rem } t : v | \mathcal{J}[x \rightarrow d] \rangle \quad -\quad t_{x \rightarrow e} \quad \rightarrow \quad \langle \emptyset | \mathcal{J}[x \rightarrow d] | D^v_{\mathcal{K}[x \rightarrow e]} \rangle
\]

\[
= \begin{cases} 
\langle \emptyset | \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \rangle & \text{if } t_{x \rightarrow e} = d \\
\text{and } v \equiv u, \\
\langle \emptyset | \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]}[x \rightarrow d] \rangle & \text{otherwise.}
\end{cases}
\]

Matching this transition with transition 5.3, we find that \( \mathcal{K} \) and \( \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \) may only differ with respect to the denotation of \( x \). Now, if \( x^\mathcal{J} = t_{x \rightarrow e} \) and \( u \equiv v \), then \( x^\mathcal{J} \notin |\mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]}|_u \) and \( x^\mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} = \infty \). Since \( \mathcal{K} \) and \( \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \) must have the same universes, we also have \( x^\mathcal{J} ^\mathcal{J} \notin |\mathcal{K}|_u \). Combining this with the requirement about \( x^\mathcal{K} \) gives us \( x^\mathcal{K} = \infty \). Putting this together yields that in this case \( \mathcal{K} = \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \).

If, however, \( x^\mathcal{J} \neq t_{x \rightarrow e} \), then \( x^\mathcal{J} \in |\mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]}|_u \) and \( x^\mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} = x^\mathcal{J} \). But, since \( \mathcal{K} \) and \( \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \) have the same universes, we also have \( x^\mathcal{J} \in |\mathcal{K}|_u \). Combining this again with the requirement about \( x^\mathcal{K} \) gives us \( x^\mathcal{K} = x^\mathcal{J} \). This yields for the second time that \( \mathcal{K} = \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \).

We see that in both cases we have \( \mathcal{K} = \mathcal{J}D^v_{\mathcal{K}[x \rightarrow e]} \). Furthermore, we have \( d' = \infty \) or \( d' = d \).

On the other hand:

\[
\langle \text{rem } t[x:=y] : v | \mathcal{J}[y \rightarrow d] \rangle \quad -\quad t_{x \rightarrow e} \quad \rightarrow \quad \langle \emptyset | \mathcal{J}[y \rightarrow d] | D^v_{\mathcal{K}[x \rightarrow y]} \rangle
\]

\[
= \begin{cases} 
\langle \emptyset | \mathcal{J}D^v_{\mathcal{K}[x \rightarrow y]} \rangle & \text{if } t_{x \rightarrow e} = d \\
\text{and } v \equiv u, \\
\langle \emptyset | \mathcal{J}D^v_{\mathcal{K}[x \rightarrow y]}[y \rightarrow d] \rangle & \text{otherwise.}
\end{cases}
\]

This is what we hoped to find.

If \( t_{x \rightarrow e} \notin |\mathcal{J}[x \rightarrow d]|_v \), then we have that:

\[
\langle \text{rem } t : v | \mathcal{J}[x \rightarrow d] \rangle \quad -\quad t_{x \rightarrow e} \quad \rightarrow \quad \langle \emptyset | \mathcal{J}[x \rightarrow d] \rangle.
\]

Matching this transition with transition 5.3, and using the restriction about \( x^\mathcal{K} \), it is easy to verify that \( \mathcal{K} = \mathcal{J} \). Moreover, we see that \( d' = d \).

On the other hand:

\[
\langle \text{rem } t[x:=y] : v | \mathcal{J}[y \rightarrow d] \rangle \quad -\quad t_{x \rightarrow e} \quad \rightarrow \quad \langle \emptyset | \mathcal{J}[y \rightarrow d] \rangle,
\]

which we hoped to find.
This settles the base case.

Now, suppose that \( \#(\mathcal{U}) = n + 1 \). In particular, suppose that \( \mathcal{U} = \mathcal{V}, \mathcal{V} \). The induction hypothesis will be applied to \( \mathcal{V} \) and \( \{ \mathcal{V} \} \). But first, assume that:

\[
(V, \mathcal{V} \mid \mathcal{J}[x \mapsto d]) \xrightarrow{\mathcal{I}[x \mapsto e]} (\emptyset \mid \mathcal{K}[x \mapsto d']) ,
\]

where \( x^\mathcal{K} \) meets the requirements.

According to the operational semantics, there must exist a \( \Sigma \)-interpretation \( \mathcal{K}'[x \mapsto d''] \) such that:

\[
(V, \mathcal{V} \mid \mathcal{J}[x \mapsto d]) \xrightarrow{\mathcal{I}[x \mapsto e]} (\mathcal{V} \mid \mathcal{K}'[x \mapsto d'']) \xrightarrow{\mathcal{I}[x \mapsto e]} (\emptyset \mid \mathcal{K}[x \mapsto d']).
\]

In order to be allowed to apply the induction hypothesis to the first part of this transition, we set:

\[
x^\mathcal{K}' = \begin{cases} 
x^\mathcal{J} & \text{if } x^\mathcal{J} \in |\mathcal{K}'|_u, \\
\infty & \text{otherwise.}
\end{cases}
\]

To be able to apply the induction hypothesis to the second part of the transition we must show that:

\[
x^\mathcal{K} = \begin{cases} 
x^\mathcal{K}' & \text{if } x^\mathcal{K}' \in |\mathcal{K}|_u, \\
\infty & \text{otherwise.}
\end{cases}
\]

Now, if \( x^\mathcal{K}' \in |\mathcal{K}|_u \), then we must have that \( x^\mathcal{K}' \neq \infty \), which is only possible if \( x^\mathcal{K}' = x^\mathcal{J} \). Then, by definition of \( x^\mathcal{K} \), we have that \( x^\mathcal{K} = x^\mathcal{J} = x^\mathcal{K}' \).

If \( x^\mathcal{K}' \notin |\mathcal{K}|_u \), then we have two possibilities: either \( x^\mathcal{K} = x^\mathcal{J} \notin |\mathcal{K}|_u \), or \( x^\mathcal{K} = \infty \). In the first case, \( x^\mathcal{K} = \infty \). In the second case, we have \( x^\mathcal{J} \notin |\mathcal{K}'|_u \), from which we infer, using Lemma 5.3.3, that \( x^\mathcal{J} \notin |\mathcal{K}|_u \). Again, this yields \( x^\mathcal{K} = \infty \). So, \( x^\mathcal{K} \) meets the requirements.

Applying the induction hypothesis twice, and the fact that \( (V, \mathcal{V})[x := y] = V[x := y], \mathcal{V}[x := y] \), yields:

\[
(V[x := y], \mathcal{V}[x := y] \mid \mathcal{J}[y \mapsto d]) \xrightarrow{\mathcal{I}[y \mapsto e]} (\mathcal{V}[x := y] \mid \mathcal{K}'[y \mapsto d'']) \xrightarrow{\mathcal{I}[y \mapsto e]} (\emptyset \mid \mathcal{K}[y \mapsto d']).
\]

This establishes the base case of the induction over \( \text{COMP} (\mathcal{U}) \).

**Induction step:** \( \text{COMP} (\mathcal{U}) = m + 1 \)

Again, we use induction over \( \#(\mathcal{U}) \). The base case is \( \#(\mathcal{U}) = 1 \). This implies that \( \mathcal{U} = \{ \text{new } z : v \text{ with } \mathcal{V} \} \), with \( \text{COMP} (\mathcal{V}) = m \). We will apply the induction hypothesis to \( \mathcal{V} \). We consider three different cases for \( z \):
5.3. Adequacy results

- $z \not\equiv x$ and $z \not\equiv y$. In this case we have $(\text{new } z : v \text{ with } \mathcal{V})[x := y] = \text{new } z : v \text{ with } \mathcal{V}[x := y]$. Suppose that the following transition is possible:

$$
\langle \text{new } z : v \text{ with } \mathcal{V} \mid \mathcal{J}[x \rightarrow d] \rangle \xrightarrow{\mathcal{I}[x \rightarrow e]} \langle \emptyset \mid \mathcal{K}[x \rightarrow d'] \rangle,
$$

such that $x$ satisfies the requirements. Then, according to our operational semantics, and taking into account that $z \not\equiv x$:

$$
\langle \mathcal{V} \mid \mathcal{J}[x \rightarrow d] \mathcal{E}_v'[z \rightarrow e'] \rangle \xrightarrow{\mathcal{I}[x \rightarrow e] \mathcal{E}_v'[z \rightarrow e']} \langle \emptyset \mid \mathcal{K}'[x \rightarrow d'] \rangle,
$$

for some $\mathcal{K}'$ such that $\mathcal{K} = \mathcal{K}'[z \rightarrow e'']$ where $e'' = z \mathcal{J}[x \rightarrow d]$ if $z \mathcal{J}[x \rightarrow d] \in [\mathcal{K}[x \rightarrow d']][v]$ and $e'' = \infty$ otherwise.

Since $\mathcal{K}$ and $\mathcal{K}'$ must have the same universes and may only differ with respect to the denotation of $z$, we find that $x \mathcal{K}' = x \mathcal{K}$. We also have that $x \mathcal{J} \mathcal{E}_v'[z \rightarrow e'] = x \mathcal{J}$. Combining these findings yields that $x \mathcal{K}'$ satisfies the requirements for applying the induction hypothesis to transition 5.4. Applying the following properties:

$$
\mathcal{J}[x \rightarrow d] \mathcal{E}_v'[z \rightarrow e'] = \mathcal{J} \mathcal{E}_v'[z \rightarrow e'][x \rightarrow d],
$$

$$
\mathcal{J}[y \rightarrow d] \mathcal{E}_v'[z \rightarrow e'] = \mathcal{J} \mathcal{E}_v'[z \rightarrow e'][y \rightarrow d],
$$

and using the induction hypothesis, we infer:

$$
\langle \mathcal{V}[x := y] \mid \mathcal{J}[y \rightarrow d] \mathcal{E}_v'[z \rightarrow e'] \rangle \xrightarrow{\mathcal{I}[y \rightarrow e] \mathcal{E}_v'[z \rightarrow e']} \langle \emptyset \mid \mathcal{K}'[y \rightarrow d'] \rangle,
$$

From this we derive:

$$
\langle \text{new } z : v \text{ with } \mathcal{V}[x := y] \mid \mathcal{J}[y \rightarrow d] \rangle \xrightarrow{\mathcal{I}[y \rightarrow e]} \langle \emptyset \mid \mathcal{K}'[y \rightarrow d'][z \rightarrow e'''] \rangle,
$$

where $e''' = z \mathcal{J}[y \rightarrow d]$ if $z \mathcal{J}[y \rightarrow d] \in [\mathcal{K}'[y \rightarrow d']][v]$, and $e''' = \infty$ otherwise. Since this implies that $e''' = e''$, and since $y \not\equiv z$ and $\mathcal{K}'[z \rightarrow e''] = \mathcal{K}$, it follows that $\mathcal{K}'[y \rightarrow d'][z \rightarrow e''''] = \mathcal{K}[y \rightarrow d']$, which settles this case.

- $z \not\equiv x$ and $z \equiv y$. Here, the substitution $[x := y]$ yields:

$$
\text{(new } z : v \text{ with } \mathcal{V})[x := y] \equiv \text{new } z' : v \text{ with } \mathcal{V}[z := z'][x := y],
$$

where $z'$ is a fresh variable. Just like in the preceding case, suppose that the following transition is possible:

$$
\langle \text{new } z : v \text{ with } \mathcal{V} \mid \mathcal{J}[x \rightarrow d] \rangle \xrightarrow{\mathcal{I}[x \rightarrow e]} \langle \emptyset \mid \mathcal{K}[x \rightarrow d'] \rangle,
$$

(5.5)
such that $x^K$ meets the requirements. Then, according to our operational semantics, and taking into account that $z \not= x$:

$$
\langle V \mid J[x \mapsto x]E_v^x[z \mapsto e'] \rangle 
\overset{T[z \mapsto e]E_v^x[z \mapsto e']}{\longrightarrow} 
\langle \emptyset \mid K'[x \mapsto d'] \rangle,
$$

for some $K'$ such that $K = K'[z \mapsto e'']$ where $e''$ is defined in the same way as in the preceding case. Using the fact that $z \not= x$, the last transition can be rewritten as follows:

$$
\langle V \mid J[x \mapsto d]E_v^x[z \mapsto e'] \rangle 
\overset{T[z \mapsto e]E_v^x[z \mapsto e']}{\longrightarrow} 
\langle \emptyset \mid K[x \mapsto d'][z \mapsto z^{K'}] \rangle.
$$

(5.6)

From $K = K'[z \mapsto e'']$ and $z \not= x$ we deduce that $x^{K[x \mapsto d']} = e''$. Since $zJ[x \mapsto d]E_v^x = zJ[x \mapsto d']$, this implies that $z^{K[x \mapsto d']}$ meets the requirements for an application of the induction hypothesis to transition 5.6:

$$
\langle V[z := z'] \mid J[x \mapsto d]E_v^x[z' \mapsto e'] \rangle 
\overset{T[z \mapsto e]E_v^x[z' \mapsto e']}{\longrightarrow} 
\langle \emptyset \mid K[x \mapsto d'][z' \mapsto z^{K'}] \rangle.
$$

Now, we want to apply the induction hypothesis a second time, but then with respect to $x$. Since $z' \not= x$ and $z' \not= y$, we find that $xJ^e_v[z \mapsto e'] = xJ$ and $x^{K[z' \mapsto z^{K'}]} = x^K$, from which we infer that $x^{K[z' \mapsto z^{K'}]}$ meets the requirements. Again using the fact that $z' \not= x$ and $z' \not= y$, and applying the induction hypothesis, yields:

$$
\langle V[z := z'][x := y] \mid J[y \mapsto d]E_v^y[z' \mapsto e'] \rangle 
\overset{T[y \mapsto e]E_v^y[z' \mapsto e']}{\longrightarrow} 
\langle \emptyset \mid K[y \mapsto d'][z' \mapsto z^{K'}] \rangle.
$$

But this implies:

$$
\langle \text{new } z' : v \text{ with } V[z := z'][x := y] \mid J[y \mapsto d] \rangle 
\overset{T[y \mapsto e]}{\longrightarrow} 
\langle \emptyset \mid K[y \mapsto d'][z' \mapsto z^{K'}][z' \mapsto e''] \rangle
$$

$$
= \langle \emptyset \mid K[z' \mapsto e''][y \mapsto d'] \rangle,
$$

such that $e''' = z'J[y \mapsto d]$ if $z'J[y \mapsto d] \in [K[y \mapsto d'][z' \mapsto z^{K'}]]_v$ and $e''' = \infty$ otherwise. This implies $e''' = z'J$ if $z'J \in [K]_v$ and $e''' = \infty$ otherwise.

We are done if we can show that $K[z' \mapsto e''] = K$, or equivalently, that $z^{K} = e'''$. But this follows from an application of Lemma 5.3.7 to transition 5.5, using the fact that $z'$ is a fresh variable, and using our findings about $e'''$.

- $z \equiv x$. This case is trivial, for we have $(\text{new } x : u \text{ with } V)[x := y] = \text{new } x : u \text{ with } V$.
This settles the base case.

For the induction step \( \#(\mathcal{U}) = n + 1 \), we reason as follows. Let \( \mathcal{W} \) be the set of all updates \( W \in \mathcal{U} \) such that \( \text{COMP}(\{W\}) = m + 1 \) and let \( \mathcal{V} = \mathcal{U} \setminus \mathcal{W} \). Then, clearly, \( \text{COMP}(\mathcal{V}) \leq m \) and \( \#(\mathcal{V}) \leq n \). Moreover, \( \mathcal{W} \) can be written as \( \mathcal{W} = \{W_1, \ldots, W_k\} \), where \( W_i \equiv \text{new} \ x_i : u_i \) with \( W_i \) for any \( i \) such that \( 1 \leq i \leq k \). Now, after checking the conditions for \( x^{K'} \) in the relevant intermediary states \( K' \) (using Lemma 5.3.3 just like in the base case of \( \text{COMP}(\mathcal{U}) = 0 \)), the induction hypothesis can be applied to \( \mathcal{V} \). With respect to the sets \( \{W_1\}, \ldots, \{W_k\} \), we may apply the base case we just proved (\( \text{COMP}(\mathcal{U}) = m + 1 \) and \( \#(\mathcal{U}) = 1 \)).

This completes the proof of Lemma 5.3.10.  

The following theorem states that an extension variable in an extension update really is a bound variable: renaming it into a fresh variable results in an operationally equivalent update.

**5.3.11 Theorem Conversion**

Let \( I, J \) and \( K \) be \( \Sigma \)-interpretations, \( \mathcal{U} \subseteq_{\text{fn}} \text{UPD} \), \( x, y \in \text{VAR}_u \) and \( u \in \text{DSORT} \) such that \( y \notin \text{FV}(\mathcal{U}) \), and:

\[
\langle \text{new} \ x : u \text{ with } \mathcal{U} \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle,
\]

then:

\[
\langle \text{new} \ y : u \text{ with } \mathcal{U}[x := y] \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle.
\]

**Proof** Suppose that we have the following transition:

\[
\langle \text{new} \ x : u \text{ with } \mathcal{U} \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle. \tag{5.7}
\]

Then, according to our operational semantics, a transition of the following form exists:

\[
\langle \mathcal{U} \mid J E_d^u[x \rightarrow d] \rangle \xrightarrow{IE_d^{x \rightarrow d}} \langle \emptyset \mid K' \rangle,
\]

such that \( K = K'[x \rightarrow d'] \), where \( d' = x^J \) if \( x^J \in |K'|_u \) and \( d' = \infty \) otherwise. The last transition can be rewritten as:

\[
\langle \mathcal{U} \mid J E_d^u[x \rightarrow d] \rangle \xrightarrow{IE_d^{x \rightarrow d}} \langle \emptyset \mid K[x \rightarrow x^{K'}] \rangle.
\]

Since \( x^K = d' \) and \( x^J E_d^u = x^J \), it follows that \( x^K \) satisfies the requirements for an application of Lemma 5.3.10 (variable change), which results in:

\[
\langle \mathcal{U}[x := y] \mid J E_d^u[y \rightarrow d] \rangle \xrightarrow{IE_d^{y \rightarrow d}} \langle \emptyset \mid K[y \rightarrow x^{K'}] \rangle.
\]

This implies:

\[
\langle \text{new} \ y : u \text{ with } \mathcal{U}[x := y] \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K[y \rightarrow x^{K'}][y \rightarrow d''] \rangle = \langle \emptyset \mid K[y \rightarrow d''] \rangle,
\]

where \( d'' = x^J \) if \( x^J \in |K[y \rightarrow x^{K'}]|_u \) (= \( |K|_u \)) and \( d'' = \infty \) otherwise. We are done if we can show that \( K[y \rightarrow d''] = K \). This is indeed the case, for since \( y \) is a fresh variable in transition 5.7, we have by Lemma 5.3.7 that \( y^K = d'' \), from which the claim follows.  

5.3.12 Lemma Weakening
Let \( I, J \) and \( K \) be \( \Sigma \)-interpretations, and let \( U, V \subseteq \text{fin UPD} \) be such that \( U \cap V = \emptyset \). Then:
\[
\langle U \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle,
\]
if and only if:
\[
\langle U \cup V \mid J \rangle \xrightarrow{I} \langle V \mid K \rangle.
\]

Proof Suppose that the following transition is possible:
\[
\langle U \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle.
\]
Let \( U = \{U_1, \ldots, U_n\} \), then there must exist a series of transitions:
\[
\langle U_1, \{U_2, \ldots, U_n\} \mid J \rangle \xrightarrow{I} \langle U_2, \{U_3, \ldots, U_n\} \mid J_1 \rangle \\
\vspace{1em}
\vdots \hspace{2em} \vdots \\
\xrightarrow{I} \langle U_n \mid J_{n-1} \rangle \\
\xrightarrow{I} \langle \emptyset \mid K \rangle.
\]
(5.8)
Since \( U_i \notin V \) for all \( i = 1, \ldots, n \), the following also is a series of transitions:
\[
\langle U_1, \{U_2, \ldots, U_n\} \cup V \mid J \rangle \xrightarrow{I} \langle U_2, \{U_3, \ldots, U_n\} \cup V \mid J_1 \rangle \\
\vspace{1em}
\vdots \hspace{2em} \vdots \\
\xrightarrow{I} \langle U_n, V \mid J_{n-1} \rangle \\
\xrightarrow{I} \langle V \mid K \rangle.
\]
(5.9)
Conversely, if the transitions 5.9 are possible, then the transitions 5.8 are possible as well, since, according to the semantics, the effect of an individual update \( U_i \) upon a state does not depend on the updates in its context, in particular the updates in \( V \). ~ \Box

5.3.13 Remark If, in the lemma above, both \( U \) and \( V \) would contain a syntactically equal update \( \hat{U} \), then \( \hat{U} \) would be 'absorbed' in the union \( U \cup V \). Therefore, it is demanded that the intersection of \( U \) and \( V \) be empty. On the other hand, if \( U \) contains an update of the form new \( x : u \) with \( W \) and \( V \) contains the update new \( y : u \) with \( W[x := y] \) for some fresh variable \( y \notin \text{fv}(W) \), then these extension updates are considered to be different, for they are syntactically different. Semantically, however, these updates have the same behaviour (cf. Theorem 5.3.11 below). In practice, these updates are likely to be inconsistent (for instance, if \( W \) contains the update \( s := x \)).

If two sets of updates \( U \) and \( V \) have no common extension updates, they can be 'sequenced': runs of \( U \cup V \) lead to the same final states as runs of \( U \) followed by runs of \( V \). Local function updates or contraction updates which are in the
intersection \( \mathcal{U} \cap \mathcal{V} \) are absorbed in \( \mathcal{U} \cup \mathcal{V} \). Extension updates, however, may not be absorbed since performing the same extension update twice leads to a different state then performing it once: every time an extension update is executed, a new element is created.

5.3.14 **Lemma Sequencing**

Let \( \mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3 \) and \( \mathcal{I} \) be \( \Sigma \)-interpretations, and let \( \mathcal{U}, \mathcal{V} \subseteq \text{fin} \ \text{UPD} \) be such that \( \mathcal{U} \cap \mathcal{V} \) does not contain any extension updates. Moreover, let:

\[
\langle \mathcal{U} \mid \mathcal{J}_1 \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J}_2 \rangle \quad \text{and} \quad \langle \mathcal{V} \mid \mathcal{J}_2 \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J}_3 \rangle,
\]

then:

\[
\langle \mathcal{U} \cup \mathcal{V} \mid \mathcal{J}_1 \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J}_3 \rangle.
\]

**Proof** The proof is by induction on the number \( \#(\mathcal{V}) \) of updates in \( \mathcal{V} \).

**Base case:** \( \#(\mathcal{V}) \leq 1 \)

Note, that if \( \#(\mathcal{V}) = 0 \), the lemma trivially holds. So, suppose that \( \#(\mathcal{V}) = 1 \). If \( \mathcal{V} = \{\text{new } x : u \text{ with } \mathcal{W}\} \), then, by the preconditions of the lemma, we know that \( \text{new } x : u \text{ with } \mathcal{W} \notin \mathcal{U} \). For this case, the lemma easily follows by applying Lemma 5.3.12 (weakening) to the first of the given transitions, and then using the second one. This means that two cases remain to be considered: \( \mathcal{V} \) contains a local function data update or a contraction update. So, we have the following three cases:

- \( \mathcal{V} = \{c := t\} \). Just like the following, more interesting case.

- \( \mathcal{V} = \{fs := t\} \). The only interesting case is if \( \mathcal{U} \) contains the update \( fs := t \). Now, since:

\[
\langle \mathcal{U} \mid \mathcal{J}_1 \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J}_2 \rangle,
\]

and by Lemma 5.3.5, \( \mathcal{J}_2 \) can be written as:

\[
\mathcal{J}_2 = \mathcal{J}_1 O_1 \ldots O_j [f[s^T \mapsto t^T]] O_{j+1} \ldots O_{j+k}.
\]

If we take \( \mathcal{U} \setminus \{fs := t\} \) and apply the remaining updates in the same order as in the given transition for \( \mathcal{U} \), we get:

\[
\langle \mathcal{U} \setminus \{fs := t\} \mid \mathcal{J}_1 \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J}_1 O_1 \ldots O_{j+k} \rangle.
\]

From the fact that a transition for \( \mathcal{V} \) starting in \( \mathcal{J}_2 \) is given, it follows that \( [f[s^T \mapsto t^T]] \) is defined on \( \mathcal{J}_2 \). But then \( [f[s^T \mapsto t^T]] \) is also defined on \( \mathcal{J}_1 O_1 \ldots O_{j+k} \), since both states have the same universes. Applying
the Weakening Lemma to the last transition, and using the fact we just found, we find:

\[ \langle \mathcal{U} \cup \{fs := t\} \mid \mathcal{J}_1 \rangle \xrightarrow{\tau} \langle fs := t \mid \mathcal{J}_1 O_1 \ldots O_{j+k} \rangle \]
\[ \xrightarrow{\tau} \langle \emptyset \mid \mathcal{J}_1 O_1 \ldots O_{j+k}[f[s^\tau \vdash t^\tau]] \rangle. \]

On the other hand, from the second given transition for \( \mathcal{V} \) we derive:

\[ \mathcal{J}_3 = \mathcal{J}_1 O_1 \ldots O_j[f[s^\tau \vdash t^\tau]] O_{j+1} \ldots O_{j+k}[f[s^\tau \vdash t^\tau]]. \]

The lemma now follows for this case, since:

\[ \mathcal{J}_1 O_1 \ldots O_j[f[s^\tau \vdash t^\tau]] O_{j+1} \ldots O_{j+k}[f[s^\tau \vdash t^\tau]] \]

and:

\[ \mathcal{J}_1 O_1 \ldots O_{j+k}[f[s^\tau \vdash t^\tau]] \]

are equal (the second update \([f[s^\tau \vdash t^\tau]]\) overrides the first one and makes it superfluous).

- \( \mathcal{V} = \{ \text{rem } t : u \} \). Again, the only interesting case is if \( \mathcal{U} \) contains \( \text{rem } t : u \).

The corresponding operator now is \( \Delta_{t^\tau}^u \). Here, the lemma directly follows, since applying the update \( \text{rem } t : u \) to \( \mathcal{J}_2 \) yields:

\[ \mathcal{J}_3 = \mathcal{J}_2 = \mathcal{J}_1 O_1 \ldots O_j \Delta_{t^\tau}^u O_{j+1} \ldots O_{j+k}, \]

for \( t^\tau \) cannot be removed twice.

**Induction step:** \( \#(\mathcal{V}) = n + 1 \)

Using the induction hypothesis, the lemma almost directly follows. This completes the proof of Lemma 5.3.14.

### 5.3.15 Lemma Universe enlarging

Let \( \mathcal{I}, \mathcal{J} \) and \( \mathcal{K} \) be \( \Sigma \)-interpretations, and \( \mathcal{U} \subseteq \text{fin } \text{UPD} \) such that:

\[ \langle \mathcal{U} \mid \mathcal{J} \rangle \xrightarrow{\tau} \langle \emptyset \mid \mathcal{K} \rangle. \]

Then, for any \( u \in \text{DSORT} \) and \( d \) such that \( d \) is available with respect to \( \mathcal{K} \) and such that \( \text{E}_d^u \) is defined on \( \mathcal{I} \), and for any \( x \in \text{VAR}_u \) such that \( x \notin \text{FV}(\mathcal{U}) \):

\[ \langle \mathcal{U} \mid J \text{E}_d^u[x \mapsto d] \rangle \xrightarrow{\tau J \text{E}_d^u[x \mapsto d]} \langle \emptyset \mid K \text{E}_d^u[x \mapsto d] \rangle. \]

**Proof** The proof is by induction on the complexity \( \text{COMP}(\mathcal{U}) \) of \( \mathcal{U} \) using Corollary 4.2.22.

**Base case:** \( \text{COMP}(\mathcal{U}) = 0 \)

We prove the base case by induction on the number of updates \( \#(\mathcal{U}) \) in \( \mathcal{U} \). The base case is \( \#(\mathcal{U}) = 1 \) (if \( \#(\mathcal{U}) = 0 \), the lemma trivially holds). We have the following cases:
5.3. Adequacy results

- \( \mathcal{U} = \{ c := t \} \), where \( c \) is a individual constant or a variable. Since \( x \notin \text{Fv}(\mathcal{U}) \), we infer that \( c \neq x \). According to the operational semantics we have:

\[
\langle c := t \mid \mathcal{J} \rangle \xrightarrow{\tau} \langle \emptyset \mid \mathcal{J}[c \leftarrow t] \rangle.
\]

On the other hand:

\[
\langle c := t \mid \mathcal{J}E^n_d[x \leftarrow d] \rangle \\
\xrightarrow{IE^n_d[x \leftarrow d]} \langle \emptyset \mid \mathcal{J}E^n_d[x \leftarrow d][c \leftarrow tIE^n_d[x \leftarrow d]] \rangle \\
= \langle \emptyset \mid \mathcal{J}[c \leftarrow t]E^n_d[x \leftarrow d] \rangle,
\]

since \( c \neq x \) and by Corollary 4.2.22. This proves this case.

- \( \mathcal{U} = \{ fs := t \} \). Similar as the preceding case.

- \( \mathcal{U} = \{ \text{rem } t : v \} \). According to our operational semantics we have:

\[
\langle \text{rem } t : v \mid \mathcal{J} \rangle \xrightarrow{\tau} \begin{cases} 
\langle \emptyset \mid \mathcal{J}D^n_{t^x} \rangle & \text{if } t^x \in |\mathcal{J}|_v, \\
\langle \emptyset \mid \mathcal{J} \rangle & \text{otherwise.}
\end{cases}
\]

On the other hand using the fact that \( tIE^n_d[x \leftarrow d] = t^x \) by Corollary 4.2.22, and the fact that \( d \neq t^x \) since \( E^n_d \) is defined on \( I \), we infer:

\[
\langle \text{rem } t : v \mid \mathcal{J}E^n_d[x \leftarrow d] \rangle \\
\xrightarrow{IE^n_d[x \leftarrow d]} \begin{cases} 
\langle \emptyset \mid \mathcal{J}E^n_d[x \leftarrow d]D^n_{t^z} \rangle & \text{if } t^x \in |\mathcal{J}E^n_d[x \leftarrow d]|_v, \\
\langle \emptyset \mid \mathcal{J}E^n_d[x \leftarrow d] \rangle & \text{otherwise.}
\end{cases}
\]

\[
= \begin{cases} 
\langle \emptyset \mid \mathcal{J}D^n_{t^z}E^n_d[x \leftarrow d] \rangle & \text{if } t^x \in |\mathcal{J}|_v, \\
\langle \emptyset \mid \mathcal{J}E^n_d[x \leftarrow d] \rangle & \text{otherwise,}
\end{cases}
\]

which establishes this case.

The induction step \#(\mathcal{U}) = n + 1 is straightforward.

Induction step: \( \text{COMP}(\mathcal{U}) = m + 1 \)

Again we use induction over \#(\mathcal{U}). The base case is \#(\mathcal{U}) = 1. This implies that \( \mathcal{U} = \{ \text{new } z : v \text{ with } \mathcal{V} \} \), with \( \text{COMP}(\mathcal{V}) = m \). We will apply the induction hypothesis to \( \mathcal{V} \). This is only allowed, however, if \( x \neq z \). But, by applying the Conversion Theorem, we can always ensure this to be the case.

So, suppose that indeed \( z \neq x \), and that the following transition is possible:

\[
\langle \text{new } z : v \mid \mathcal{V} \mid \mathcal{J} \rangle \xrightarrow{\tau} \langle \emptyset \mid \mathcal{K} \rangle.
\]

Then we also should have:

\[
\langle \mathcal{V} \mid \mathcal{J}E^n_c[z \leftarrow e] \rangle \xrightarrow{TE^n_c[z \leftarrow e]} \langle \emptyset \mid \mathcal{K}' \rangle,
\]
such that $K = K'[z \mapsto e'']$ where $e''$ satisfies:

$$e'' = \begin{cases} 
  z^J & \text{if } z^J \in |K'|_u, \\
  \infty & \text{otherwise}.
\end{cases}$$

Since $d$ is available with respect to $K$, we infer from inspecting the transition rule of extension updates that $d$ is also available with respect to $K'$. But then Proposition 5.3.2 implies that $d$ is also available with respect to $JE^u_e[z \mapsto e]$, from which we deduce that $d \neq e$. Combining this fact with the assumption of the lemma that $E^u_d$ is defined on $I$, gives us that $E^u_d$ is also defined on $JE^u_e[z \mapsto e]$. This means we may apply the induction hypothesis to infer:

$$\langle V \mid J E^u_e[z \mapsto e]E^u_d[x \mapsto d] \rangle \xrightarrow{IE^u_e[z \mapsto e]E^u_d[x \mapsto d]} \langle \emptyset \mid K'E^u_d[x \mapsto d] \rangle.$$  

Since $z \neq x$ this implies:

$$\langle V \mid J E^u_d[x \mapsto d]E^u_e[z \mapsto e] \rangle \xrightarrow{IE^u_d[x \mapsto d]E^u_e[z \mapsto e]} \langle \emptyset \mid K'E^u_d[x \mapsto d] \rangle.$$  

From which we derive:

$$\langle \text{new } z : v \text{ with } V \mid J E^u_d[x \mapsto d] \rangle \xrightarrow{IE^u_d[x \mapsto d]} \langle \emptyset \mid K'E^u_d[x \mapsto d][z \mapsto e'''] \rangle,$$

where $e'''$ is defined by:

$$e''' = \begin{cases} 
  z^JE^u_d[x \mapsto d] & \text{if } z^JE^u_d[x \mapsto d] \in |K'E^u_d[x \mapsto d]|_u, \\
  \infty & \text{otherwise}.
\end{cases}$$

Since $z^JE^u_d[x \mapsto d] = z^J$ and $z^J \neq d$, it follows that $e''' = e''$ and:

$$K'E^u_d[x \mapsto d][z \mapsto e'''] = K'[z \mapsto e'']E^u_d[x \mapsto d] = K'E^u_d[x \mapsto d],$$

which completes this case.

The induction case $\#(U) = n + 1$, is again left to the reader. This means we are done with the proof of Lemma 5.3.15.

The following lemma is variant of the preceding one. It can be proven using the same technique. Therefore, we omit the proof.

**5.3.16 Lemma** Let $I$, $J$ and $K$ be $\Sigma$-interpretations, and $U \subseteq \text{fin } UPD$ such that:

$$\langle U \mid J \rangle \xrightarrow{I} \langle \emptyset \mid K \rangle,$$

then for any $u \in DSORT$ and $d$ such that $d$ is not available w.r.t. $J$ and such that $E^u_d$ is defined on $I$, and for any $x \in \text{VAR}_u$ such that $x \notin \text{Fv}(U)$:

$$\langle U \mid J \rangle \xrightarrow{IE^u_d[x \mapsto d]} \langle \emptyset \mid K \rangle.$$
5.3. Adequacy results

The next lemma is about the scope of updates. It tells us that updates outside the scope of an extension update may be put into the scope of that extension update, provided that no name clashes can occur.

At this point the reader should recall Convention 5.2.32, according to which the expression new \( x : u \) with \( U, V \) is used to denote the union of the sets \{new \( x : u \) with \( U \}\} and \( V \) whenever these sets are disjoint.

5.3.17 Lemma Rescoping

Let new \( x : u \) with \( U, V \subseteq \text{fin UPD} \), and let \( I, J \) and \( K \) be \( \Sigma \)-interpretations such that:

\[
\langle \text{new } x : u \text{ with } U, V | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle,
\]

then for any \( y \notin \text{Fv}(U) \cup \text{Fv}(V) \):

\[
\langle \text{new } y : u \text{ with } U[x := y] \cup V | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle.
\]

Proof Suppose that the following transition is possible:

\[
\langle \text{new } x : u \text{ with } U, V | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle.
\]

Let \( V = \{V_1, \ldots, V_n\} \), then there must exist a series of transitions:

\[
\langle \text{new } x : u \text{ with } U, \{V_1, \ldots, V_n\} | J \rangle
\]

\[
\xrightarrow{I} \langle \text{new } x : u \text{ with } U, \{V_2, \ldots, V_n\} | J_1 \rangle
\]

\[
\vdots
\]

\[
\xrightarrow{I} \langle \text{new } x : u \text{ with } U, \{V_{m+1}, \ldots, V_n\} | J_m \rangle
\]

\[
\xrightarrow{I} \langle \{V_{m+1}, \ldots, V_n\} | J_{m+1} \rangle
\]

\[
\vdots
\]

\[
\xrightarrow{I} \langle V_n | J_n \rangle
\]

\[
\xrightarrow{I} \langle \emptyset | K \rangle.
\]

By Lemma 5.3.12 (weakening) and Theorem 5.3.11 (conversion) we have for some fresh variable \( y \notin \text{Fv}(U) \cup \text{Fv}(V) \):

\[
\langle \text{new } y : u \text{ with } U[x := y], \{V_{m+1}, \ldots, V_n\} | J_m \rangle
\]

\[
\xrightarrow{I} \langle \{V_{m+1}, \ldots, V_n\} | J_{m+1} \rangle.
\]

According to our semantics this transition is possible if the following transition exists:

\[
\langle U[x := y] | J_m E_d[y \rightarrow d] \rangle
\]

\[
\xrightarrow{I E_d[y \rightarrow d]} \langle \{V_{m+1}, \ldots, V_n\} | K' \rangle,
\]

such that \( J_{m+1} = K'[y \rightarrow e] \), where \( e \) meets the requirements posed by the operational semantics of an extension update. This equality implies that \( K' = J_{m+1}[y \rightarrow y'K'] \).

(5.10)
Chapter 5. Syntax and semantics of Evolving Algebras

Now, by applying the Weakening Lemma to transition 5.10 we get:

\[ \langle \{ V_1, \ldots, V_m \} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{F}} \langle \emptyset \mid \mathcal{J}_m \rangle. \]  
(5.12)

From the same transition it also follows that:

\[ \langle \{ V_{m+1}, \ldots, V_n \} \mid \mathcal{J}_{m+1} \rangle \xrightarrow{\mathcal{F}} \langle \emptyset \mid \mathcal{K} \rangle. \]  
(5.13)

Since \( y \notin \text{Fv}(\mathcal{V}) \), and since for \( d \) we have that \( \mathcal{E}^u_d \) is defined on \( \mathcal{I} \), and that \( d \) is available with respect to \( \mathcal{J}_m \), we may apply Lemma 5.3.15 (universe enlarging) to transition 5.12. This yields:

\[ \langle \{ V_1, \ldots, V_m \} \mid \mathcal{J} \mathcal{E}^u_d[y \rightarrow d] \rangle \xrightarrow{\mathcal{I} \mathcal{E}^u_d[y \rightarrow d]} \langle \emptyset \mid \mathcal{J}_m \mathcal{E}^u_d[y \rightarrow d] \rangle. \]

Furthermore, since \( y \notin \text{Fv}(\mathcal{V}) \), and since we have that \( \mathcal{E}^u_d \) is defined on \( \mathcal{I} \), and that \( d \) is not available with respect to \( \mathcal{J}_{m+1} \), we may apply Lemma 5.3.16 and Lemma 5.3.9 to transition 5.13. This results in:

\[ \langle \{ V_{m+1}, \ldots, V_n \} \mid \mathcal{J}_{m+1}[y \rightarrow y'] \rangle \xrightarrow{\mathcal{I} \mathcal{E}^u_d[y \rightarrow d]} \langle \emptyset \mid \mathcal{K}[y \rightarrow e'] \rangle, \]

where \( e' \) meets the conclusions of the latter lemma.

Combining the findings of above with transition 5.11, and using the Sequencing Lemma yields:

\[ \langle \mathcal{U}[x := y] \cup \mathcal{V} \mid \mathcal{J} \mathcal{E}^u_d[y \rightarrow d] \rangle \xrightarrow{\mathcal{I} \mathcal{E}^u_d[y \rightarrow d]} \langle \emptyset \mid \mathcal{K}[y \rightarrow e'] \rangle. \]

From this we derive:

\[ \langle \text{new } y : u \text{ with } \mathcal{U}[x := y] \cup \mathcal{V} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{F}} \langle \emptyset \mid \mathcal{K}[y \rightarrow e'][y \rightarrow e''] \rangle \]
\[ = \langle \emptyset \mid \mathcal{K}[y \rightarrow e''] \rangle. \]

where:

\[ e'' = \begin{cases} y^\mathcal{J} & \text{if } y^\mathcal{J} \in |\mathcal{K}[y \rightarrow e']|_u = |\mathcal{K}|_u, \\
\infty & \text{otherwise}. \end{cases} \]

When we apply Lemma 5.3.7 to the original transition 5.10, we get:

\[ y^\mathcal{K} = \begin{cases} y^\mathcal{J} & \text{if } y^\mathcal{J} \in |\mathcal{K}|_u, \\
\infty & \text{otherwise}, \end{cases} \]

which implies \( \mathcal{K}[y \rightarrow e''] = \mathcal{K} \). But then we have proven Lemma 5.3.17.  

A consequence of this lemma is that sets of updates can be brought into a normal form. The format of this normal form looks like the well known prenex normal form.

5.3.18 Definition Let \( \mathcal{U} \subseteq_{\text{fn}} \text{UPD} \) be a set of updates. Then \( \mathcal{U} \) is said to be in normal form, if either \( \mathcal{U} \) does not contain any extension updates, or \( \mathcal{U} \) has the format:

\[ \{ \text{new } x_1 : u_1 \text{ with } \{ \text{new } x_2 : u_2 \text{ with } \{ \ldots \{ \text{new } x_n : u_n \text{ with } \mathcal{V} \} \ldots \} \} \}, \]

where \( n \geq 1 \) and \( \mathcal{V} \) does not contain any extension updates.
5.3. Adequacy results

5.3.19 Theorem Normal form
There exists an operation $\text{Norm}(\_)$ such that for any set $U \subseteq \text{fin UPD}$ any result of $\text{Norm}(U)$ is a normal form of $U$. Moreover, if for some $\Sigma$-interpretations $I$, $J$ and $K$:

$$\langle U | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle,$$

then for any result $V$ of $\text{Norm}(U)$:

$$\langle V | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle.$$

Proof Define the operation $\text{Norm}(\_)$ as follows:

$$\text{Norm}(V) = V \quad \text{if } V \text{ is in normal form},$$

$$\text{Norm}(\text{new } x : u \text{ with } U, V)$$

$$= \{\text{new } y : u \text{ with } \text{Norm}(U|x:=y| U \cup V)\},$$

where $y \notin \text{Fv}(U) \cup \text{Fv}(V)$. It is clear that $\text{Norm}(U)$ is in normal form for any $U \subseteq \text{fin UPD}$. Moreover, using the Rescoping Lemma, it is easy to show by induction on the number of necessary rescoping steps that the operational behaviour of any result $V$ of the operation $\text{Norm}(U)$ is as claimed by this theorem. ■

5.3.20 Remark The operation $\text{Norm}(\_)$ is not a function. Its result depends on the order in which the rescoping process of the extension updates in $U$ takes place, and which extension variables are chosen.

Consider the following set $U \subseteq \text{fin UPD}$:

$$U = \{\text{new } x : u \text{ with new } y : v \text{ with } W\}.$$

Clearly, $U$ is in normal form, whenever $W$ is. But, note that:

$$V = \{\text{new } y : v \text{ with new } x : u \text{ with } W\}$$

can never by a result of $\text{Norm}(U)$: only $U$ itself is. So, we cannot conclude from the preceding theorem that:

$$\langle V | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle,$$

whenever:

$$\langle U | J \rangle \xrightarrow{I} \langle \emptyset | K \rangle.$$

This contrasts with our intuition that both $U$ and $V$ are normal forms of $V$. However, in the next proposition we will see that $V$ has the same operational behaviour as $U$. 
5.3.21 Proposition Let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be $\Sigma$-interpretations. Moreover, let for some $\mathcal{U} \subseteq \text{fin} \ \text{UPD}$:

$$\langle \text{new } x_1 : u_1 \text{ with } \ldots \text{ new } x_n : u_n \text{ with } \mathcal{U} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{K} \rangle,$$

then, for any permutation $i_1 \ldots i_n$ of $1 \ldots n$:

$$\langle \text{new } x_{i_1} : u_{i_1} \text{ with } \ldots \text{ new } x_{i_n} : u_{i_n} \text{ with } \mathcal{U} \mid \mathcal{J} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{K} \rangle.$$

Proof Firstly, note that by the Conversion Theorem (5.3.11) we may assume the extension variables $x_1, \ldots, x_n$ to be different. By applying the rule for extension updates we infer from the given transition that the following transition exists:

$$\langle \mathcal{U} \mid \mathcal{J} \mathcal{E}_{d_1}^{u_1}[x_1 \mapsto d_1] \ldots \mathcal{E}_{d_n}^{u_n}[x_n \mapsto d_n] \rangle$$

$$\xrightarrow{\mathcal{I} \mathcal{E}_{d_1}^{u_1}[x_1 \mapsto d_1] \ldots \mathcal{E}_{d_n}^{u_n}[x_n \mapsto d_n]} \langle \emptyset \mid \mathcal{K}' \rangle,$$

for some $\Sigma$-interpretation $\mathcal{K}'$ such that $\mathcal{K} = \mathcal{K}'[x_n \mapsto e_n] \ldots [x_1 \mapsto e_1]$ for some appropriate $e_1 \ldots e_n$. Since all the extension variables are different, we have:

$$\mathcal{J} \mathcal{E}_{d_1}^{u_1}[x_1 \mapsto d_1] \ldots \mathcal{E}_{d_n}^{u_n}[x_n \mapsto d_n] = \mathcal{J} \mathcal{E}_{d_{i_1}}^{u_{i_1}}[x_{i_1} \mapsto d_{i_1}] \ldots \mathcal{E}_{d_{i_n}}^{u_{i_n}}[x_{i_n} \mapsto d_{i_n}],$$

$$\mathcal{I} \mathcal{E}_{d_1}^{u_1}[x_1 \mapsto d_1] \ldots \mathcal{E}_{d_n}^{u_n}[x_n \mapsto d_n] = \mathcal{I} \mathcal{E}_{d_{i_1}}^{u_{i_1}}[x_{i_1} \mapsto d_{i_1}] \ldots \mathcal{E}_{d_{i_n}}^{u_{i_n}}[x_{i_n} \mapsto d_{i_n}],$$

$$\mathcal{K}'[x_{i_n} \mapsto e_{i_n}] \ldots [x_{i_1} \mapsto e_{i_1}] = \mathcal{K}'[x_{i_n} \mapsto e_{i_n}] \ldots [x_{i_1} \mapsto e_{i_1}].$$

From these facts the proposition follows. \end{proof}

5.3.22 Convention If $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ are $\Sigma$-interpretations such that $|\mathcal{K}|_u \subseteq |\mathcal{I}|_u$ for all $u \in \text{SORT}$, then $F : \mathcal{I} \to \mathcal{J}$ denotes a family of functions:

$$\{F_u : |\mathcal{I}|_u \to |\mathcal{J}|_u \mid u \in \text{SORT}\},$$

and $F \mid \mathcal{K}$ denotes the family of restricted functions:

$$\{F_u \mid |\mathcal{K}|_u : |\mathcal{K}|_u \to |\mathcal{J}|_u \mid u \in \text{SORT}\}.$$
5.3. Adequacy results

5.3.23 Theorem Isomorphism

Let $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{J}_1$, $\mathcal{J}_2$ and $\mathcal{K}_1$ be $\Sigma$-interpretations. Furthermore, let $F : \mathcal{J}_1 \to \mathcal{J}_2$ and $G : \mathcal{I}_1 \to \mathcal{I}_2$ be isomorphisms such that $F_u(d) = G_u(e) \in |\mathcal{J}_2|_u \cap |\mathcal{I}_2|_u$ if and only if $d = e \in |\mathcal{J}_1|_u \cap |\mathcal{I}_1|_u$, for all $d$ and $e$, and all $u \in \text{SORT}$. Moreover, let $\mathcal{U} \subseteq \text{RN \cup \text{PD}}$, and:

$$(\mathcal{U} \ | \ \mathcal{J}_1) \xrightarrow{\mathcal{I}_1} (\emptyset \ | \ \mathcal{K}_1).$$

Then there exists a $\Sigma$-interpretation $\mathcal{K}_2$ and an isomorphism $H : \mathcal{K}_1 \to \mathcal{K}_2$, such that:

$$(\mathcal{U} \ | \ \mathcal{J}_2) \xrightarrow{\mathcal{I}_2} (\emptyset \ | \ \mathcal{K}_2),$$

and such that $H_u(d) = F_u(e) \in |\mathcal{K}_2|_u \cap |\mathcal{J}_2|_u$ if and only if $d = e \in |\mathcal{K}_1|_u \cap |\mathcal{J}_1|_u$, and such that $H_u(d) = G_u(e) \in |\mathcal{K}_2|_u \cap |\mathcal{I}_2|_u$ if and only if $d = e \in |\mathcal{K}_1|_u \cap |\mathcal{I}_1|_u$, for all $d$ and $e$, and all $u \in \text{SORT}$.

Proof The proof is by induction on the complexity $\text{COMP} (\mathcal{U})$ of $\mathcal{U}$.

Base case: $\text{COMP} (\mathcal{U}) = 0$

By Lemmas 5.3.5 and 5.3.6, $\mathcal{K}_1$ can be written as $\mathcal{K}_1 = \mathcal{J}_1 O_1 \ldots O_n$, where each $O_i$ corresponds to a simple update $U$ which is evaluated with respect to $\mathcal{I}_1$. Define $O'_i$ to be the operator corresponding to $U$ but now interpreted with respect to the evaluation state $\mathcal{I}_2$. We are going to show by induction on $n$ that the $\mathcal{K}_2$ we are looking for is $\mathcal{K}_2 = \mathcal{J}_2 O'_1 \ldots O'_n$, and that $F \ | \ \mathcal{K}_1 : \mathcal{K}_1 \to \mathcal{K}_2$ is the isomorphism $H$. In particular, we will construct a series $F^0, \ldots, F^n$ of isomorphisms:

$$F^j : \mathcal{J}_1 O_1 \ldots O_j \to \mathcal{J}_2 O'_1 \ldots O'_j,$$

with the property that for all $u \in \text{SORT}$, and all $d$ and $e$:

$$d = e \in |\mathcal{J}_1 O_1 \ldots O_j|_u \cap |\mathcal{I}_1|_u \Leftrightarrow F^j_u(d) = G_u(e) \in |\mathcal{J}_2 O'_1 \ldots O'_j|_u \cap |\mathcal{I}_2|_u, \quad (5.14)$$

and such that:

$$F^j = F \ | \ \mathcal{J}_1 O_1 \ldots O_j. \quad (5.15)$$

For the base case, $n = 0$, we define $H = F^0$ (=$F$). In that case, $H$ clearly meets the conditions 5.14 and 5.15, and $H$ also meets the properties of $H$ mentioned in the theorem, which proves the base case.

With respect to the induction step, suppose that $F^j$ is an isomorphism meeting the appropriate conditions. We now have to define $F^{j+1}$ in such way that $F^{j+1}$ meets this conditions too. Consider the following cases for $O_{j+1}$:

- $O_{j+1} = [e \mapsto t_{\mathcal{I}_1}]$. Just like the following, more interesting case.
• \( O_{j+1} = [f [s_{\mathcal{T}_1} \mapsto t_{\mathcal{T}_1}]] \) where \( s \in \text{TERM}^w \) and \( t \in \text{TERM}_v \). So, we have that \( O'_{j+1} = [f [s_{\mathcal{T}_2} \mapsto t_{\mathcal{T}_2}]] \). Since \( O_{j+1} \) is defined on \( \mathcal{J}_1 O_1 \ldots O_j \), it follows that \( s_{\mathcal{T}_1} \in |\mathcal{J}_1 O_1 \ldots O_j^w| \cap |T_1|^w \). From the induction hypothesis we then may derive that \((F^j)^w s_{\mathcal{T}_1} = G^w s_{\mathcal{T}_1} \in |\mathcal{J}_2 O'_1 \ldots O'_j|^w \). Using the induction hypothesis again, we infer for \( t \) that \( F^j_v(t_{\mathcal{T}_1}) = G_v(t_{\mathcal{T}_1}) \in |\mathcal{J}_2 O'_1 \ldots O'_j|_v \) if \( t_{\mathcal{T}_1} \in |\mathcal{J}_1 O_1 \ldots O_j|_v \). Furthermore, since \( F^j \) and \( G \) are isomorphisms, \( F^j_v(t_{\mathcal{T}_1}) = G_v(t_{\mathcal{T}_1}) = \infty \) if \( t_{\mathcal{T}_1} = \infty \). As another consequence of the fact that \( G \) is an isomorphism, we have that \( G^w s_{\mathcal{T}_1} = s_{\mathcal{T}_2} \), and \( G_v(t_{\mathcal{T}_1}) = t_{\mathcal{T}_2} \). Combining all these facts, we derive that \( O'_{j+1} \) is defined on \( \mathcal{J}_2 O'_1 \ldots O'_j \).

Now, define \( F^{j+1} = F^j \). This is reasonable, since the universes of \( \mathcal{J}_1 O_1 \ldots O_j \) and \( \mathcal{J}_2 O'_1 \ldots O'_j \) are not changed by \( O_{j+1} \) and \( O'_{j+1} \), respectively. Using the properties we derived above, we infer:

\[
F^{j+1}_v(f_{\mathcal{J}_1 O_1 \ldots O_{j+1}} s_{\mathcal{T}_1}) = F^j_v(f_{\mathcal{J}_1 O_1 \ldots O_{j+1}} s_{\mathcal{T}_1}) = F^j_v(t_{\mathcal{T}_1}) = G_v(t_{\mathcal{T}_1}) = t_{\mathcal{T}_2} = f_{\mathcal{J}_2 O'_1 \ldots O'_{j+1}} s_{\mathcal{T}_2} = f_{\mathcal{J}_2 O'_1 \ldots O'_{j+1}} G^w s_{\mathcal{T}_1} = f_{\mathcal{J}_2 O'_1 \ldots O'_{j+1}} (F^j)^w s_{\mathcal{T}_1} = f_{\mathcal{J}_2 O'_1 \ldots O'_{j+1}} (F^{j+1})^w s_{\mathcal{T}_1}.
\]

Since the interpretation of \( f \) has not been changed by \( O_{j+1} \) except for the argument \( s_{\mathcal{T}_1} \), we infer from the findings above and from the induction hypothesis that \( F^{j+1} \) is an isomorphism which meets conditions 5.14 and 5.15.

• \( O_{j+1} = D^v_{\mathcal{T}_1} \) for some \( t \in \text{TERM}_v \). Here, we have that \( O'_{j+1} = D^v_{\mathcal{T}_2} \). Analogously as in the preceding case, we may derive that \( O'_{j+1} \) is defined on \( \mathcal{J}_2 O'_1 \ldots O'_j \) whenever \( O_{j+1} \) is defined on \( \mathcal{J}_1 O_1 \ldots O_j \). The latter being the case, it follows that \( t_{\mathcal{T}_1} \) is removed from \( |\mathcal{J}_1 O_1 \ldots O_j|_v \) and that \( F^j_v(t_{\mathcal{T}_1}) = t_{\mathcal{T}_2} \) is removed from \( |\mathcal{J}_2 O'_1 \ldots O'_j|_v \).

Moreover, from the induction hypothesis and the fact that \( G \) is an isomorphism, it follows that for any term \( s \in \text{TERM}_v \):

\[
s_{\mathcal{J}_1 O_1 \ldots O_j} = t_{\mathcal{T}_1} \iff s_{\mathcal{J}_2 O'_1 \ldots O'_j} = t_{\mathcal{T}_2}.
\]

This means that the removal of \( t_{\mathcal{T}_1} \) and \( t_{\mathcal{T}_2} \), respectively, results in setting the same corresponding terms in \( \mathcal{J}_1 O_1 \ldots O_{j+1} \) and \( \mathcal{J}_2 O'_1 \ldots O'_{j+1} \) to \( \infty \). In other words, \( O_{j+1} \) and \( O'_{j+1} \) have the same ‘side effects’.

Define \( F^{j+1} = F^j \upharpoonright \mathcal{J}_1 O_1 \ldots O_{j+1} \). Then it follows from our findings that \( F^{j+1} \) is an isomorphism meeting the appropriate conditions.
5.3. Adequacy results

Since each $F^{i+1}$ equals $F^i$ or is a restriction of it, and since $F^0 = F$, we infer that $H = F^n$ satisfies the properties for $H$. This completes the proof for the base case $\text{COMP}(U) = 0$.

**Induction step:** $\text{COMP}(U) = m + 1$

Assume the theorem holds if $\text{COMP}(U) = m$. For simplicity we will assume that $U$ contains exactly one extension update $U$ such that $\text{COMP}(\{U\}) = m + 1$ (the general case can easily be proven by induction). Let $U \equiv \text{new } x : v$ with $V$. Since:

$$
\langle U \mid J_1 \rangle \xrightarrow{T_1} \langle \emptyset \mid K_1 \rangle,
$$

there exist disjoint sets $U_1$ and $U_2$ not containing $U$ such that $U_1 \cup U_2 \cup \{U\} = U$, and such that $\text{COMP}(U_1) \leq m$, $\text{COMP}(U_2) \leq m$, and:

$$
\begin{align*}
\langle U_1 \cup U_2 \cup \{U\} \mid J_1 \rangle & \xrightarrow{T_1} \langle U, U_2 \mid K'_1 \rangle \\
& \xrightarrow{T_1} \langle U_2 \mid K''_1 \rangle \\
& \xrightarrow{T_1} \langle \emptyset \mid K_1 \rangle.
\end{align*}
$$

From this we infer that the following transition must exist:

$$
\langle V \mid K'_1 E^v_d[x \rightarrow d] \rangle \xrightarrow{T_{1} E^v_d[x \rightarrow d]} \langle \emptyset \mid K''''_1 \rangle,
$$

where $K''''_1 = K''''_1[x \rightarrow d']$ such that:

$$
\begin{align*}
d' &= \begin{cases} 
  x & \text{if } x \in |K''''_1|_v, \\
  \infty & \text{otherwise}.
\end{cases}
\end{align*}
$$

On the other hand, since $\text{COMP}(U_1) \leq m$, we have by the induction hypothesis:

$$
\langle U_1 \mid J_2 \rangle \xrightarrow{T_2} \langle \emptyset \mid K'_2 \rangle,
$$

for some $K'_2$ isomorphic to $K'_1$. Let $H : K'_1 \rightarrow K'_2$ be the isomorphism satisfying the properties mentioned in the theorem.

Now, consider $K'_2 E^v_c[x \rightarrow e]$ and $T_{2} E^v_c[x \rightarrow e]$ for some $e$ such that $e$ is available with respect to $K'_2$, and such that $E^v_c$ is defined on $T_2$. Then it is clear that we can construct isomorphisms:

$$
\begin{align*}
H' : & \quad K'_1 E^v_d[x \rightarrow d] \rightarrow K'_2 E^v_c[x \rightarrow e], \\
G' : & \quad T_{1} E^v_d[x \rightarrow d] \rightarrow T_{2} E^v_c[x \rightarrow e],
\end{align*}
$$

such that $H' \mid K'_1 = H$, $H'_v(d) = e$, $G' \mid T_1 = G$, and $G'_v(d) = e$. These $H'$ and $G'$ are in accordance with the conditions of the theorem. This means we may apply the induction hypothesis to transition 5.16 to obtain:

$$
\langle V \mid K'_2 E^v_c[x \rightarrow e] \rangle \xrightarrow{T_{2} E^v_c[x \rightarrow e]} \langle \emptyset \mid K''''_2 \rangle,
$$
for some $\mathcal{K}'_2''$ isomorphic to $\mathcal{K}'_2''$. Let $H'' : \mathcal{K}'_1'' \rightarrow \mathcal{K}'_2''$ be the isomorphism satisfying the properties mentioned in the theorem. From the latter transition we infer:

$$\langle U \mid \mathcal{K}'_2 \rangle \xrightarrow{\mathcal{I}_2} \langle \emptyset \mid \mathcal{K}'_2'' \rangle,$$

where $\mathcal{K}'_2'' = \mathcal{K}'_2''[x \mapsto e']$ such that:

$$e' = \begin{cases} x^{\mathcal{K}'_2} & \text{if } x^{\mathcal{K}'_2} \in |\mathcal{K}'_2''|_{\mathcal{V}}, \\ \infty & \text{otherwise.} \end{cases}$$

As $H'' : \mathcal{K}'_1'' \rightarrow \mathcal{K}'_2''$ is in accordance with the properties of the theorem, we have

$$H''_v(a) = H'_v(b) \in |\mathcal{K}'_2''|_v \cap |\mathcal{K}'_2 E'_v[x \mapsto e']|_v \text{ iff } a = b \in |\mathcal{K}'_1''|_v \cap |\mathcal{K}' E'_v[x \mapsto d]|_v.$$ 

Using the definition of $H'$, the fact that $x^{\mathcal{K}'_1} \neq d$, and the fact that $H$ is an isomorphism, we infer that $H'_v(x^{\mathcal{K}'_1}) = H_v(x^{\mathcal{K}'_1}) = x^{\mathcal{K}'_2}$. Putting all things together, we deduce that $x^{\mathcal{K}'_1} \in |\mathcal{K}'_1''|_v$ iff $H'_v(x^{\mathcal{K}'_1}) = x^{\mathcal{K}'_2} \in |\mathcal{K}'_2''|_v$, and, consequently, that $H''_v(d') = e'$. But this implies that $H'' : \mathcal{K}'_1'' \rightarrow \mathcal{K}'_2''$ is also an isomorphism. This $H''$ obeys the properties mentioned in the theorem, in this case with respect to $H$ and $G$.

We already had that:

$$\langle U_2 \mid \mathcal{K}'_1'' \rangle \xrightarrow{\mathcal{I}_1} \langle \emptyset \mid \mathcal{K}_1 \rangle,$$

so, again, we may apply the induction hypothesis. This yields:

$$\langle U_2 \mid \mathcal{K}'_2'' \rangle \xrightarrow{\mathcal{I}_2} \langle \emptyset \mid \mathcal{K}_2 \rangle,$$

such that an isomorphism $H'''' : \mathcal{K}_1 \rightarrow \mathcal{K}_2$ exists having the properties mentioned in the theorem with respect to $H''$ and $G$. Finally, putting all relevant transitions together and applying the Sequencing Lemma (5.3.14) yields:

$$\langle U \mid \mathcal{J}_2 \rangle \xrightarrow{\mathcal{I}_2} \langle \emptyset \mid \mathcal{K}_2 \rangle.$$ 

Using Lemma 5.3.3, it is easy to show that $H''''$ also satisfies the properties mentioned in the theorem with respect to $F$ and $G$. This completes the proof of Theorem 5.3.23.

5.3.24 REMARK Recall Remark 5.2.35 where it was said that the choice of a reserve element in an extend operator is nondeterministic. However, Theorem 5.3.23 ensures that different choices of reserve elements result in isomorphic states.

5.3.25 LEMMA Let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be $\Sigma$-interpretations. Moreover, let $U \subseteq \text{fin UPD}$ be in normal form, and let Cons$(U, I)$. Moreover, let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be $\Sigma$-interpretations such that:

$$\langle U \mid \mathcal{I} \rangle \xrightarrow{\mathcal{F}} \langle \emptyset \mid \mathcal{J} \rangle \text{ and } \langle U \mid \mathcal{I} \rangle \xrightarrow{\mathcal{F}} \langle \emptyset \mid \mathcal{K} \rangle.$$ 

Then $\mathcal{J}$ and $\mathcal{K}$ are isomorphic.
Proof The proof is by induction on the complexity COMP(\mathcal{U}) of \mathcal{U}. In order to have an applicable induction hypothesis, we will prove the slightly stronger claim that there exists an isomorphism \( H : \mathcal{J} \to \mathcal{K} \) such that \( H_u(e) = e \in |\mathcal{K}|_u \cap |\mathcal{I}|_u \) iff \( d = e \in |\mathcal{J}|_u \cap |\mathcal{I}|_u \), for all \( u \in \text{SORT} \).

For the base case \( \text{COMP}(\mathcal{U}) = 0 \), we have that \( \mathcal{U} \) does not contain any extension updates. It then follows from the given transitions that:

\[
\begin{align*}
\mathcal{J} & = IO_1 \ldots O_m, \\
\mathcal{K} & = IO_1 \ldots O_m,
\end{align*}
\]

where \( m \leq \#(\mathcal{U}) \), and where where each \( O_j \) (\( 1 \leq j \leq m \)) is a local modification operator or a remove operator, and where \( i_1 \ldots i_m \) is a permutation of \( 1 \ldots m \).

From the fact that \( \text{Cons}(\mathcal{U}, \mathcal{I}) \) holds, and the fact that \( \mathcal{U} \) does not contain extension updates, we deduce that \( \mathcal{C}(\mathcal{U}) \equiv \mathcal{C}^+(\mathcal{U}) \) and that \( \mathcal{I} \models \mathcal{C}^+(\mathcal{U}) \).

Inspection of the definition of \( \mathcal{C}^+(\mathcal{U}) \) (Definition 5.2.13) learns us that the order of application of the operators \( O_j \) is immaterial. (This can be seen by checking that for each pair \( O_j O_k \) of operators (\( 1 \leq j, k \leq m \)) we have that \( \mathcal{J} O_j O_k = \mathcal{J} O_k O_j \) for any \( \Sigma \)-interpretation \( \mathcal{J} \) such that \( O_j \) and \( O_k \) are defined on it.) It also follows from this definition that any order of applications of these operators is defined on \( \mathcal{I} \). From this observations we infer that \( \mathcal{J} = \mathcal{K} \), and that the identity isomorphism \( id : \mathcal{J} \to \mathcal{K} \) satisfies the claim made at the beginning of this proof. This means that the lemma holds for the base case.

Assume, for the induction step, that \( \text{COMP}(\mathcal{U}) = n + 1 \). Now, since \( \mathcal{U} \) is in normal form, \( \mathcal{U} \) can be written as:

\[
\mathcal{U} = \{ \text{new } x : u \text{ with } \mathcal{V} \},
\]

where \( \text{COMP}(\mathcal{V}) = n \) and \( \mathcal{V} \) is in normal form. By applying the rule for extension updates, we infer from the given transitions that the following transitions exist:

\[
\begin{align*}
\langle \mathcal{V} | \mathcal{I} \mathcal{E}^u_d[x \mapsto d] \rangle & \xrightarrow{\mathcal{I} \mathcal{E}^u_d[x \mapsto d]} \langle \emptyset | \mathcal{J}' \rangle, \\
\langle \mathcal{V} | \mathcal{I} \mathcal{E}^u_e[x \mapsto e] \rangle & \xrightarrow{\mathcal{I} \mathcal{E}^u_e[x \mapsto e]} \langle \emptyset | \mathcal{K}' \rangle,
\end{align*}
\]

for some \( d \) and \( e \) available with respect to \( \mathcal{I} \), and some \( \Sigma \)-interpretations \( \mathcal{J}' \) and \( \mathcal{K}' \), such that \( \mathcal{J} = \mathcal{J}'[x \mapsto d'] \) and \( \mathcal{K} = \mathcal{K}'[x \mapsto e'] \) where:

\[
d' = \begin{cases} 
  x^\mathcal{I} & \text{if } x^\mathcal{I} \in |\mathcal{J}'|_u, \\
  \infty & \text{otherwise},
\end{cases}
\]

and:

\[
e' = \begin{cases} 
  x^\mathcal{I} & \text{if } x^\mathcal{I} \in |\mathcal{K}'|_u, \\
  \infty & \text{otherwise}.
\end{cases}
\]

Consider the isomorphism \( F : \mathcal{I} \mathcal{E}^u_e[x \mapsto e] \to \mathcal{I} \mathcal{E}^u_d[x \mapsto d] \), defined by \( F \upharpoonright \mathcal{I} = id \) and \( F_u(e) = d \) where \( id \) denotes the identity isomorphism. Then by applying
Theorem 5.3.23 (isomorphism) to transition 5.18 we infer that there exists a Σ-interpretation \( \mathcal{K}' \) and an isomorphism \( G : \mathcal{K} \to \mathcal{K}'' \) such that:
\[
\langle \mathcal{V} | \mathcal{I}E_\mathcal{I}^\mathcal{K}[x \to d] \rangle \xrightarrow{\mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d]} \langle \emptyset | \mathcal{K}'' \rangle,
\]
(5.19)
and \( G_u(a) = F_u(b) \in |\mathcal{K}''|_u \cap |\mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d]|_u \) iff \( a = b \in |\mathcal{K}'|_u \cap |\mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to e]|_u \).
Since \( x^I \neq e \), we have that \( F(x^I) = x^I \). Combining this with what we just found yields that \( G_u(x^I) = x^I \in |\mathcal{K}''|_u \) iff \( x^I \in |\mathcal{K}'|_u \).

If we can show that \( \mathcal{K}'' \) and \( \mathcal{I}' \) are isomorphic too, we would be almost done. In order to do so we would like to apply the induction hypothesis to transitions 5.17 and 5.19. But in that case we need that \( \text{Cons}(\mathcal{V}, \mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d]) \).

Since \( \text{Cons}(\mathcal{U}, \mathcal{I}) \) it follows by Theorem 5.2.20 that:
\[
\mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d] | \mathcal{U}^+ = C^+(\mathcal{U}^+),
\]
where \( x \) is the vector of all extension variables of \( \mathcal{U} \), the first component of which is \( x \), and where \( u \) is the vector of all sort names associated with \( x \), the first component of which is \( u \). Define \( x = (x, y_1, \ldots, y_n) \) and \( y = (y_1, \ldots, y_n) \), define \( u = (u_1, \ldots, u_n) \) and \( v = (v_1, \ldots, v_n) \), and, finally, define \( d = (d, e_1, \ldots, e_n) \) and \( e = (e_1, \ldots, e_n) \). Then, our consistency condition can be rewritten as:
\[
\mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d]|\mathcal{IE}_\mathcal{I}^\mathcal{K}[y \to e] \models C^+(\mathcal{V}^y),
\]
using Definition 5.2.12 and the fact that \( \mathcal{U} \) is in normal form. But then, by Theorem 5.2.20 again, we have that \( \text{Cons}(\mathcal{V}, \mathcal{IE}_\mathcal{I}^\mathcal{K}[x \to d]) \). Applying the induction hypothesis to transitions 5.17 and 5.19 yields that \( \mathcal{K}'' \) is isomorphic to \( \mathcal{I}' \). In particular it follows that there exists an isomorphism \( H : \mathcal{K}'' \to \mathcal{I}' \) obeying the claim at the beginning of this proof. In particular this claim implies that \( H_u(x^I) = x^I \in |\mathcal{I}'|_u \) iff \( x^I \in |\mathcal{K}''|_u \).

We already saw that \( \mathcal{K}' \) and \( \mathcal{K}'' \) are isomorphic, so we infer that \( \mathcal{I}' \) and \( \mathcal{K}' \) are isomorphic as well. Finally, the conditions on \( d' \) and \( e' \), the fact that \( G_u(a^I) = x^I \in |\mathcal{K}''|_u \) iff \( x^I \in |\mathcal{K}'|_u \), and the fact that \( H_u(x^I) = x^I \in |\mathcal{I}'|_u \) iff \( x^I \in |\mathcal{K}''|_u \), imply that \( H \circ G \) is an isomorphism between \( \mathcal{K} \) and \( \mathcal{I} \). This completes the proof of Lemma 5.3.25.

5.3.26 Theorem Soundness \( \xrightarrow{T} \)
Let \( I, J \) and \( K \) be Σ-interpretations, and let \( U \subseteq U \) such that \( \text{Cons}(U, I) \), and:
\[
\langle U | I \rangle \xrightarrow{T} \langle \emptyset | J \rangle \quad \text{and} \quad \langle U | I \rangle \xrightarrow{T} \langle \emptyset | K \rangle,
\]
then \( J \) and \( K \) are isomorphic.

Proof Let \( V \) be a result of \( \text{Norm}(U) \). Then by Theorem 5.3.19:
\[
\langle V | I \rangle \xrightarrow{T} \langle \emptyset | J \rangle \quad \text{and} \quad \langle V | I \rangle \xrightarrow{T} \langle \emptyset | K \rangle.
\]
Using Theorem 5.2.20 and Definitions 5.2.12 and 5.2.13 it is easy to verify that \( \text{Cons}(U, I) \) implies \( \text{Cons}(V, I) \). But then we may apply Lemma 5.3.25 to conclude that \( J \) and \( K \) are isomorphic.
5.3. Adequacy results

5.3.27 Theorem Completeness \( \xrightarrow{I} \)
Let \( I \) be a \( \Sigma \)-interpretation, and let \( \mathcal{U} \subseteq \textup{fin UPD} \) such that Cons \((\mathcal{U}, I)\) holds. Then for some \( \Sigma \)-interpretation \( J \):
\[
\langle \mathcal{U} | I \rangle \xrightarrow{I} \langle \emptyset | J \rangle.
\]

Proof By the Conversion Theorem (5.3.11) we may assume all the extension variables to be different. Furthermore, by Lemma 5.3.5 the transition:
\[
\langle \mathcal{U} | I \rangle \xrightarrow{I} \langle \emptyset | J \rangle
\]
exists if and only if \( J = I O_1 \ldots O_n \) for some operators \( O_1 \ldots O_n \). We will prove by induction over \( n \) that \( J \) is defined.

If \( n = 0 \), then clearly \( \mathcal{U} = \emptyset \), in which case the theorem holds. For the induction step, assume that \( I O_1 \ldots O_m \) is defined (\( 0 \leq m < n \)). We will show that \( O_{m+1} \) is defined on \( I O_1 \ldots O_m \). Consider the following possibilities:

- \( O_{m+1} = [c \mapsto t^K] \). Just like the following, more interesting case.

- \( O_{m+1} = \{f[s^K \mapsto t^K]\} \) for some \( s \in \text{TERM}^\mathcal{U} \) and \( t \in \text{TERM}^\mathcal{U} \). By Lemma 5.3.6, \( \kappa = \mathcal{I}E_d^\mathcal{U}[x \mapsto d] \) where \( x \) are the extension variables belonging to the extension updates in which \( f s := t \) is nested. We have to show that \( s^K \in |I O_1 \ldots O_m|^w \), and that \( t^K \in |I O_1 \ldots O_m|_u \) if \( t^K \neq \infty \).

Now, by Lemma 5.3.6 again, the operators occurring in \( E_d^\mathcal{U} \) all precede \( O_{m+1} \). This implies that \( O_{m+1} \) can only be undefined on \( I O_1 \ldots O_m \) if one of the elements of \( s^K \) or \( t^K \) is removed by a remove operator preceding \( O_{m+1} \), or if \( s^K = \infty \) for some component \( s_i \) of \( s \). We will show that both situations are impossible.

In the first situation there would be a remove operator, say \( D_{s_i}^{u'} \), preceding \( O_{m+1} \). Then \( u' \equiv u \) or \( u' \) is one of the elements of \( w \), and \( s^{K'} = s^K \) or \( s^{K'} = s_i^K \) for some component \( s_i \) of \( s \). Furthermore, \( K' = E_{d'}^u[x' \mapsto d'] \) where \( x' \) are the extension variables belonging to the extension updates in which \( r s : u' \) is nested.

From Cons \((\mathcal{U}, I)\) and from Theorem 5.2.20, we infer that \( \mathcal{I}E_d^\mathcal{U}[y \mapsto e] \models C^+(\mathcal{U}^y) \). Here, \( y \) are all extension variables in \( \mathcal{U} \) (we already assumed them to be different). These include \( x \) and \( x' \). Moreover, we assume the \( e \) to be chosen to include \( d \) and \( d' \) in the appropriate way. Let \( K'' = \mathcal{I}E_d^\mathcal{U}[y \mapsto e] \).

By Corollary 4.2.22 and the fact that the components of \( y \) which are not in \( x \) or \( x' \), cannot occur in \( s \), \( t \) or \( s \), we have that \( s^{K''} = s^K \), \( t^{K''} = t^K \), and \( s^{K''} = s^{K'} \). Combining this with the fact that \( \mathcal{I}E_d^\mathcal{U}[y \mapsto e] \models C^+(\mathcal{U}^y) \) we infer that \( s_i^K \neq s_i^{K'} \) for all components \( s_i \) of \( s \), and that \( t^K \neq s_i^{K'} \).

This contradicts our assumption about \( D_{s_i}^{u'} \).

Moreover, from the same consistency condition we infer that \( s_i^K \neq \infty \) for all components \( s_i \) of \( s \), which rules out the second situation as well. We conclude that \( O_{m+1} \) is defined on \( I O_1 \ldots O_m \).
• $O_{m+1} = D_{t^*_K}$. Just like in the preceding case we can show that $t^K \in |IO_1 \ldots O_m|_u$, which is sufficient for $O_{m+1}$ to be defined on $IO_1 \ldots O_m$.

• $O_{m+1} = E_d^u$. By Lemma 5.3.6 we have that $d \notin |IO_1 \ldots O_m|_u$. This means that $O_{m+1}$ is defined on $IO_1 \ldots O_m$.

This completes the proof. ■

The last theorems of this chapter establish soundness and completeness of the relation $\rightarrow_R$.

5.3.28 Theorem Soundness $\rightarrow_R$

Let $I$, $J$ and $K$ be $\Sigma$-interpretations, and let $R$ be a consistent, deterministic $EA$. Moreover, let:

$I \rightarrow_R J$ and $I \rightarrow_R K$,

such that $J$ and $K$ are final states. Then both runs have equal length, and $J$ and $K$ are isomorphic.

Proof. We will firstly show, that $J$ and $K$ are isomorphic, if the transitions $I \rightarrow_R J$ and $I \rightarrow_R K$ have equal length. Then, to complete the proof, we will show that, if $J$ and $K$ are final states, both transitions have equal length.

The proof of the first claim is by induction on the number $n$ of steps in the transitions. If $n = 0$, the theorem trivially holds.

For the induction step, suppose that:

$I \rightarrow_R J'$ and $I \rightarrow_R K'$,

where both transitions have the length of $m$ steps ($0 \leq m < n$) and where $J'$ and $K'$ are isomorphic. If $J'$ nor $K'$ are final states, then there must exist rules $R_1, R_2 \in R$ such that $J' \models R_1$? and $K' \models R_2$?. From the fact that $J'$ and $K'$ are isomorphic, we deduce that also $J' \models R_2$? and $K' \models R_1$?. But then $R_1 = R_2$, as $R$ is deterministic. Let $R$ be the rule $R_1 (= R_2)$.

By Theorem 5.3.27 and the fact that $R$ is consistent, we infer that the following transitions exist:

$\langle R! \mid J' \rangle \rightarrow' \langle \emptyset \mid J'' \rangle$ and $\langle R! \mid K' \rangle \rightarrow' \langle \emptyset \mid K'' \rangle$.

Then by applying Theorems 5.3.23 and 5.3.26 we deduce that $J''$ and $K''$ are isomorphic.

In order to prove the second claim above, assume that $J$ and $K$ are final states. We will show, by a reductio ad absurdum, that the two transitions in the theorem have equal length. So, without loss of generality, assume that there exists a state $K'$ such that:

$I \rightarrow R J$ and $I \rightarrow R K' \rightarrow R K$,

where $I \rightarrow R J$ and $I \rightarrow R K'$ have equal length, and where $J$ is a final state but $K'$ is not. From this we infer that there is a rule $R \in R$ such that
5.3. Adequacy results

$\mathcal{K}' \models R?$. Since the first part of the proof yields that $\mathcal{J}$ and $\mathcal{K}'$ are isomorphic, we deduce that $\mathcal{J} \models R?$. Therefore, $\mathcal{J}$ is not a final state. This contradicts our assumption, and we are done. ■

**5.3.29 Theorem Completeness** $\xrightarrow{R}$

Let $I$ be a $\Sigma$-interpretation, and let $\mathcal{R}$ be a consistent $\text{EA}$. Moreover, let $I \xrightarrow{\mathcal{R}} J$

be a run with final state $\mathcal{J}$. Then $\mathcal{J}$ is a normal final state.

**Proof** Since $\mathcal{R}$ is consistent, we have that Cons $(R!, \mathcal{K})$, whenever $\mathcal{K} \models R?$. So, by Theorem 5.3.27 there exists a state $\mathcal{K}'$ such that:

$$\langle R! \mid \mathcal{K} \rangle \xrightarrow{\mathcal{K}} \langle \emptyset \mid \mathcal{K}' \rangle,$$

whenever $\mathcal{K} \models R?$. This means that the computation can always proceed from a state $\mathcal{K}$ whenever there is a rule $R \in \mathcal{R}$ such that $\mathcal{K} \models R?$. If there is not such a rule, then obviously $\mathcal{K} \models \neg R?$ for all $R \in \mathcal{R}$. In that case, the computation halts in a normal final state. ■
Chapter 6

A proof system for Evolving Algebras

This chapter contains a Hoare-style proof system for evolving algebras. Hoare-style proof systems, also known as axiomatic semantics, can be seen as a tool to describe the semantics of programming languages. Many text books have been written on this subject, see for example [AO91a, AO91b, Dah92, Fra92].

In the area of evolving algebras not much has been done on this subject, with possible exception of [PH94]. However, in this paper another point of view is taken. The author aims at deriving partial correctness logics for deterministic sequential programming languages specified by EAs.

We will directly address the problem of developing a (partial and total) correctness logic for EAs themselves. This will turn out not be a trivial task due to the fact we are working in an EA formalism allowing for extension and contraction updates.

In Section 6.1 we present the derivation rules of the proof system. We also show the rules to be correct. However, the system will turn out to be incomplete due to the contraction update. Finally, in Section 6.2 we will give an correctness proof for the sample EA of Chapter 5.

6.1 The proof system

We will start this section by formulating and proving an important theorem. In fact, it is a generalization of Theorems 4.2.17, 4.2.21, and 4.2.23. It will be used in proving the correctness of the derivation rule for sets of simple updates, later in this section.

In accordance with its future use, the theorem will be formulated in terms of sets of simple updates. Before stating the theorem, we need some definitions. The first definition is a generalization of Definition 4.2.19. The notion of a relativization of a formula is now extended to sets of terms. Note, that in the formulae below, $\bigwedge$ has a higher precedence than $\rightarrow$.

6.1.1 Definition. Let $\varphi \in \text{FORM}$ and let $T = \{t_1, \ldots, t_n\}$ be a set of terms such that $t_i \in \text{TERM}_u$, for $i = 1, \ldots, n$. Then the relativization $\varphi^T$ of $\varphi$ with respect to $T$ is recursively defined by:

\[
\bar{t}^T = t, 
\]
\[ \overline{ff}^T = \ \overline{fr}^T = \downarrow r, \]
\[ \overline{(r = s)}^T = \ (r = s), \]
\[ \overline{Pr}^T = Pr, \]
\[ \overline{-\psi}^T = \neg \psi^T, \]
\[ \overline{(\psi \ast \chi)}^T = (\overline{\psi}^T \ast \overline{\chi}^T), \]

\[ \overline{(\forall x^v \psi)}^T = \begin{cases} (\forall x^v (\bigwedge_{i=1,\ldots,n} \neg (x^v = t_i) \rightarrow \overline{\psi}^T)) & \text{if } u_i \equiv v \text{ for some } i, \\ (\forall x^v \overline{\psi}^T) & \text{if } u_i \not\equiv v \text{ for all } i, \end{cases} \]

\[ \overline{(\exists x^v \psi)}^T = \begin{cases} (\exists x^v (\bigwedge_{i=1,\ldots,n} \neg (x^v = t_i) \rightarrow \overline{\psi}^T)) & \text{if } u_i \equiv v \text{ for some } i, \\ (\exists x^v \overline{\psi}^T) & \text{if } u_i \not\equiv v \text{ for all } i. \end{cases} \]

In defining a derivation rule for sets of simple updates we want to exploit the idea that sets of local function updates can be seen as substitutions, provided that the rules in which these sets occur are (relatively) consistent. However there is a problem, which we will illustrate in the following example.

6.1.2 Example Consider the following rule:

if \( b = c \) then
\[
\begin{align*}
f(a) &:= b \\ f(a) &:= c \\ d &:= c
\end{align*}
\]

Then, clearly, we have Cons(\{f(a) := b, f(a) := c, d := c\}, b = c), which means that the rule is consistent, although it is not very likely that someone writes it down. The problem here is that [f(a) := b, f(a) := c, d := c] is not a substitution, due to the fact that f(a) occurs twice as left-hand side. However, both \( \sigma_1 = [f(a) := b, d := c] \) and \( \sigma_2 = [f(a) := c, d := c] \) are substitutions. Moreover, both \( \sigma_1 \) and \( \sigma_2 \) are maximal subsets of \{f(a) := b, f(a) := c, d := c\} being substitutions. The choice between \( \sigma_1 \) and \( \sigma_2 \) is immaterial, since for all \( \varphi \in \text{FORM} \) we have \( b = c \models \varphi \sigma_1 \leftrightarrow \varphi \sigma_2 \).

6.1.3 Definition Let \( \mathcal{U} \subseteq \text{fin UPD} \) be a set of simple updates. Then \( \sigma_\mathcal{U} \) is called a maximal substitution with respect to \( \mathcal{U} \), if the following conditions hold:

1. \( \sigma_\mathcal{U} \) is a substitution;
2. \( \sigma_\mathcal{U} \subseteq \mathcal{U} \);

3. for all sets \( \mathcal{V} \subseteq \mathcal{U} \), if \( \sigma_\mathcal{U} \subset \mathcal{V} \), then \( \mathcal{V} \) is not a substitution.

Clearly, if \( \sigma_\mathcal{U} \) is a maximal substitution w.r.t. \( \mathcal{U} \), then \( \sigma_\mathcal{U} \) only contains local function updates. The next proposition restates the conclusion of the example above that the choice between maximal substitutions is immaterial from a logical point of view.

6.1.4 PROPOSITION Let \( \mathcal{U} \subseteq \text{fin UPD} \) be a set of simple updates, and let \( \varphi \in \text{FORM} \) be such that \( \text{Cons}(\mathcal{U}, \varphi) \). Moreover, let \( \sigma_\mathcal{U} \) and \( \tau_\mathcal{U} \) be maximal substitutions w.r.t. \( \mathcal{U} \). Then for all formulae \( \psi \in \text{FORM} \):

\[
\varphi \models \psi \sigma_\mathcal{U} \leftrightarrow \psi \tau_\mathcal{U}.
\]

PROOF Using the assumption that \( \text{Cons}(\mathcal{U}, \varphi) \) and Definition 5.2.13, it is easy to see that \( \sigma_\mathcal{U}^I = \tau_\mathcal{U}^I \) for all interpretations \( I \) such that \( I \models \varphi \). Then the proposition follows by applying the Substitution Theorem (4.2.17).  

If we recall Theorems 4.2.21 and 4.2.23, then we see that applying the operator \( D^\mathcal{U}_t \) to an interpretation \( I \) in some way corresponds to applying the substitution \( [t:=\bot] \) to a term or formula, if a certain condition is satisfied. Now, suppose that \( \mathcal{U} \) is a set of simple updates which is consistent with respect to some \( \Sigma \)-interpretation \( I \), then we would like to have a theorem which relates a set of updates to a certain substitution. However, there is a small problem, which we will explain in the following example.

6.1.5 EXAMPLE Consider the following rule:

\[
\text{if } a \neq b \text{ then } \\
\quad a := b \\
\quad \text{rem } a : u
\]

This rule clearly is consistent. Regarding the update set \( \mathcal{U} = \{a:=b, \text{rem } a : u\} \), we only have one maximal substitution \( \sigma_\mathcal{U} = [a:=b] \). The substitution related to \( \text{rem } a : u \) is \( [a:=\bot] \). Now, the union \( [a:=b, a:=\bot] \) is not a substitution. So, this union cannot be the substitution related to \( \mathcal{U} \). But what is, in this case?

If we think about why in Theorems 4.2.21 and 4.2.23 the operator \( D^\mathcal{U}_t \) corresponds to the substitution \( [t:=\bot] \), we see that \( [t:=\bot] \) is not directly related to \( D^\mathcal{U}_t \), but to its side effect that removing \( t^I \) also makes the value of \( t \) undefined. Now, in our example rule, \( a \) gets assigned the value of \( b \) at the same time that the value of \( a \) is removed. This means that there is no side effect related to \( a \), since \( a \) got already a new value. We conclude that the substitution related to \( \mathcal{U} \) must be \( [a:=b] \).

In the next definition this idea will be used in defining the substitution \( \sigma_{U,\bot} \). For all contraction updates \( \text{rem } s : u \in T_\mathcal{U} \) this substitution will contain the substitution \( s := \bot \) if and only if no term \( t \) exists such that the simple
update $s := t$ occurs in $U$. If $\sigma_U$ is a maximal substitution for $U$, then clearly $\sigma_U \cup \sigma_{U, \bot}$ is a substitution. Moreover, it is the substitution related to $U$ we were looking for.

**6.1.6 Definition** Let $U \subseteq_{\text{fin}} \text{UPD}$ solely consist of simple updates. Then the sets $T_U$ and $\sigma_{U, \bot}$ are defined by:

$$T_U = \{ s : \text{rem } s : u \in U \text{ for some } u \},$$

$$\sigma_{U, \bot} = \{ s := \bot : s \in T_U \text{ and } s := t \notin U \text{ for all terms } t \}.$$

**6.1.7 Convention** If $U \subseteq_{\text{fin}} \text{UPD}$ is a set of simple updates, consistent in some context, then $\sigma_U$ will be used to denote some unspecified substitution which is maximal w.r.t. $U$.

The convention above is justified by Proposition 6.1.4. Recall the theorem we are looking for: a generalization of Theorems 4.2.17, 4.2.21, and 4.2.23. We will now define the condition related to the conditions A, B and C of the last mentioned theorems.

**6.1.8 Definition** Let $s \in \text{XTERM}_v$, and let $T = \{ t_1, \ldots, t_n \}$ be a set of terms such that $t_i \in \text{TERM}_{u_i}$ ($1 \leq i \leq n$). Define the following sets:

- $F_s$ is the set of all function symbols $f$ occurring in $s$ such that $f \in \text{FUN}_{w, u_i}$ for some $w \in \text{SORT}^+$ and $i$ ($1 \leq i \leq n$);

- $C_s$ is the set of all constant symbols $c$ occurring in $s$ such that $c \in \text{CON}_{u_i}$ for some $i$ ($1 \leq i \leq n$);

- $X_s$ is the set of all variables $x^{u_i} \in \text{VAR}_{u_i} \cap \text{FV}(s)$ for some $i$ ($1 \leq i \leq n$);

- $F_T$ is the set of all function symbols $f \in \text{FUN}_{w, u_i}$ such that $fs \in T$ for some $w \in \text{SORT}^+$, $s \in \text{TERM}^w$, and $i$ ($1 \leq i \leq n$);

- $C_T$ is the set of all constant symbols $c \in \text{CON}_{u_i}$ such that $c \in T$ for some $i$ ($1 \leq i \leq n$);

- $X_T$ is the set of all variables $x^{u_i} \in \text{VAR}_{u_i}$ such that $x^{u_i} \in T$ for some $i$ ($1 \leq i \leq n$).

Then the remove condition $\text{Crem}(s, T)$ is defined by:

$$\bigwedge_{i=1, \ldots, n} \left( \downarrow t_i \wedge \bigwedge_{f \in F_s, f \notin F_T} \overline{\forall x(fx \neq t_i)}^T \wedge \bigwedge_{c \in C_s \cup X_s, c \notin C_T \cup X_T} \overline{c \neq t_i} \wedge \bigwedge_{f \in F_s \cap F_T} \bigwedge_{s \in \text{TERM}^w} \overline{\forall x \left( \bigwedge_{fs \notin T} \overline{fx \neq t_i} \right)}^T \right),$$
where all variables $x$ are fresh, i.e. do not occur in any $t_i$ or $s$.

Analogously, $\text{Crem}(\varphi, T)$ can be defined for $\varphi \in \text{FORM}$. In that case one has to use the sets $F_{\varphi}$, $C_{\varphi}$, and $X_{\varphi}$. The definitions of these sets can be obtained by changing ‘$s$’ into ‘$\varphi$’ everywhere.

Roughly, the conditions $\text{Crem}(s, T)$ and $\text{Crem}(\varphi, T)$, respectively, express that $s$ and $\varphi$ do not contain unwanted references to the elements of $T$. This means that removing the elements denoted by the terms in $T$ has no hidden side effects upon $s$ or $\varphi$.

6.1.9 DEFINITION Let $U \subseteq_{\text{fin}} \text{UPD}$ be a set of simple updates, and let \{\text{rem } t_1 : u_1, \ldots, \text{rem } t_n : u_n\} be the set of all contraction updates in $U$. Moreover, let $I$ be a $\Sigma$-interpretation. Then the set $D^T_{U}$ is defined by:

$$D^T_{U} = \{D^{u_1}_{t_1^T}, \ldots, D^{u_n}_{t_n^T}\}.$$

6.1.10 CONVENTION Let $I$ be a $\Sigma$-interpretation, and let $D = \{D^{u_1}_{d_1^T}, \ldots, D^{u_n}_{d_n^T}\}$ be a set of remove operators such that $d_i \neq d_j$, whenever $u_i \equiv u_j$ and $i \neq j$, and such that $d_i \in |I|_{u_i^T}$, for all $i$ and $j$ ($1 \leq i, j \leq n$). Then $ID$ abbreviates:

$$ID^{u_1}_{d_1^T}, \ldots, D^{u_n}_{d_n^T}.$$

The following proposition is straightforward. We will omit its proof.

6.1.11 PROPOSITION Let $I$ be a $\Sigma$-interpretation, and let $U \subseteq_{\text{fin}} \text{UPD}$ be a set of simple updates. If $\text{Cons}(U, I)$, then:

$$I\sigma^T_{U}D^T_{U}$$

is well defined.

Now we have come to the theorem we were looking for. The line of its proof closely follows the proofs of Theorems 4.2.21 and 4.2.23.

6.1.12 THEOREM Let $U \subseteq_{\text{fin}} \text{UPD}$ only contain simple updates, and let $I$ be a $\Sigma$-interpretation such that $\text{Cons}(U, I)$. Moreover, let $s \in \text{TERM}_u$, and let $\varphi \in \text{FORM}$. Then:

$$sI\sigma^T_{U}D^T_{U} = (s \sigma_U \cup \sigma_{u, \perp})^T,$$

$$I\sigma^T_{U}D^T_{U} \models \varphi \iff I \models \varphi \sigma_U \cup \sigma_{u, \perp}^{-T_U},$$

provided that $I \models \text{Crem}(s, U)$ and $I \models \text{Crem}(\varphi, U)$, respectively.

PROOF The proof of the first part of the theorem is by simultaneous induction on the structure of extended terms $s$ and extended function symbols $f \in X_{\text{FUN}}_{u,v}$. This can be done, for the theorem can be adapted to also
cover extended function symbols. Let the sets $F_f$, $C_f$ and $X_f$, and the remove condition $\textbf{Crem}(f, I)$ be defined as expected. Then:

$$
\text{DOM} \big( f^I \sigma_U^D U \big) = \text{DOM} \big( (f \sigma_U \cup \sigma_{U, \perp})^I \big) \cap |\mathcal{I}D_U^I|^w,
$$

$$
f^I \sigma_U^D U \mid x = (f \sigma_U \cup \sigma_{U, \perp})^I x,
$$

provided that $\mathcal{I} \models \textbf{Crem}(f, T_U)$.

So, assume that $\mathcal{I} \models \textbf{Crem}(s, T_U)$ and $\mathcal{I} \models \textbf{Crem}(f, T_U)$. Moreover, let $\mathcal{I} = \langle \mathcal{A}, \beta \rangle$. We have the following cases:

- $s \equiv \perp$. Trivial.

- $s \equiv x^u$. We consider three possibilities:
  - $x^u$ is affected by $\sigma_U$, say $[x^u := l] \in \sigma_U$. Note that this implies that $[x^u := \perp] \notin \sigma_{U, \perp}$. Since Cons $(U, \mathcal{I})$ holds, we have that $D_{(x^u)}^I \notin D_U^I$, from which we deduce that $(x^u)^I \sigma_U^D U = t^I$. On the other hand $(x^u \sigma_U \cup \sigma_{U, \perp})^I = t^I$, which proves this case.
  - $x^u$ is affected by $\sigma_{U, \perp}$. Now, $[x^u := \perp] \in \sigma_{U, \perp}$, $[x^u := \perp] \notin \sigma_U$, and $D_{(x^u)}^I \in D_U^I$. We infer that $(x^u)^I \sigma_U^D U = \beta \triangleright (x^u)^I \triangleright (x^u)^I = \perp^I = (x^u \sigma_U \cup \sigma_{U, \perp})^I$, which we had to show.
  - $x^u$ is not affected by $\sigma_U \cup \sigma_{U, \perp}$. Then $x^u$ is not affected by $\sigma_U$, and $x^u \notin T_U$. Using the fact that $\mathcal{I} \models \textbf{Crem}(x^u, T_U)$, we derive that $(x^u)^I \neq t^I$ for all $t \in T_U$. But this implies that $(x^u)^I \sigma_U^D U = (x^u)^I = (x^u \sigma_U \cup \sigma_{U, \perp})^I$.

- $s \equiv c$. Like the preceding case.

- $f \in \text{FUN}_{w, v}$. Let $f \sigma_U \cup \sigma_{U, \perp} \equiv f[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]$. Then, by Definition 4.1.11:

$$
(f \sigma_U \cup \sigma_{U, \perp})^I x = \begin{cases} 
  s_1^I & \text{if } x = r_1^I, \\
  \vdots & \\
  s_m^I & \text{if } x = r_m^I, \\
  f^I \mid x & \text{otherwise},
\end{cases}
$$

for all $x \in \text{DOM} \big( (f \sigma_U \cup \sigma_{U, \perp})^I \big)\), where:

$$
\text{DOM} \big( (f \sigma_U \cup \sigma_{U, \perp})^I \big) = \text{DOM} \big( f^I \big) \setminus \{ r_i^I \mid \sim C(i) \& \ 1 \leq i \leq m \} \cup \{ r_i^I \mid C(i) \& \ 1 \leq i \leq m \} = \text{DOM} \big( f^I \big) \setminus \{ r_i^I \mid s_i^I = \infty \& \ 1 \leq i \leq m \} \cup \{ r_i^I \mid s_i^I \neq \infty \& \ 1 \leq i \leq m \}.
$$
6.1. The proof system

The last equation follows from the fact that Cons \((\mathcal{U}, \mathcal{I})\), and the definition of the conditions \(C(i)\).

On the other hand, by Definitions 4.2.1 and 4.2.7:

\[
f^{I\sigma_u^D_u, I, D_u}_x = \begin{cases} 
e x = d \text{ and } f[d \mapsto e] \in \sigma_u^D_u, \\ f^I_x \text{ otherwise}, \\ s^I_i \text{ if } x = r^I_i, \\ \vdots \quad \vdots \\ s^I_m \text{ if } x = r^I_m, \\ f^I_x \text{ otherwise}, \end{cases}
\]

for all \(x \in \text{DOM}(f^{I\sigma_u^D_u, I, D_u})\), where:

\[
\text{DOM}(f^{I\sigma_u^D_u, I, D_u}) = (\text{DOM}(f^I) \setminus \{d \mid f[d \mapsto \infty] \in \sigma_u^D_u\}) \\
\quad \cup \{d \mid f[d \mapsto e] \in \sigma_u^D_u \text{ for some } e \neq \infty\}) \cap |ID_u|^w \setminus \{d \mid D_f^v d \in D_u^I\}.
\]

Since Cons \((\mathcal{U}, \mathcal{I})\), we are done with this case if we can show that:

\[
\{d \mid d \in |ID_u|^w \quad \& \quad D_f^v d \in D_u^I\} \\
\subseteq \{r^I_i \mid s^I_i = \infty \quad \& \quad 1 \leq i \leq m\}.
\]

This however, easily follows from the fact that \(\mathcal{I} \models \text{Crem}(f, I, d)\).

- \(f \equiv g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]\) for some \(g \in \text{XFUN}_{w, v}\), some \(r_1, \ldots, r_m \in \text{XTERM}^w\), and some \(s_1, \ldots, s_m \in \text{XTERM}_v\). In the following \(g[r_i \mapsto s_i]\) will be used to abbreviate the expression \(g[r_1 \mapsto s_1, \ldots, r_m \mapsto s_m]\). In this case, we have:

\[
g[r_i \mapsto s_i]_{\sigma_u} \cup \sigma_u \perp \equiv g_{\sigma_u} \cup \sigma_u \perp [r_i \sigma_u \cup \sigma_u \perp \mapsto s_i]_{\sigma_u} \cup \sigma_u \perp.
\]

Since all symbols occurring in \(g\), \(r_i\) and \(s_i\) also occur in \(f\), we may apply the appropriate induction hypotheses upon them. This yields:

\[
\text{DOM}(g^{I\sigma_u^D_u, I, D_u}) = \text{DOM}((g_{\sigma_u} \cup \sigma_u \perp)^I) \cap |ID_u|^w,
\]

\[
g^{I\sigma_u^D_u, I, D_u}_x = (g_{\sigma_u} \cup \sigma_u \perp)^I x,
\]

\[
r^I_i^{I\sigma_u^D_u} = (r_i \sigma_u \cup \sigma_u \perp)^I,
\]

\[
s^I_i^{I\sigma_u^D_u} = (s_i \sigma_u \cup \sigma_u \perp)^I.
\]
These equalities imply that, if the condition C(i) of Definition 4.1.11 holds with respect to \((r_i \sigma_U \cup \sigma_{U,\perp})^T\) and \((s_i \sigma_U \cup \sigma_{U,\perp})^T\) \((1 \leq i \leq m)\), it will also hold with respect to \(r_i^{\sigma_U^T D_U^T}\) and \(s_i^{\sigma_U^T D_U^T}\) \((1 \leq i \leq m)\). Taking also into account that \(\{r_i^{\sigma_U^T D_U^T} \mid C(i) \& 1 \leq i \leq m\} \subseteq |ID_U^T|^w\), and using Definition 4.1.11 again, we conclude that:

\[
\begin{align*}
\text{DOM}(f^{\sigma_U^T D_U^T}) &= \text{DOM}((f \sigma_U \cup \sigma_{U,\perp})^T) \cap |ID_U^T|^w, \\
 f^{\sigma_U^T D_U^T} x &= (f \sigma_U \cup \sigma_{U,\perp})^T x.
\end{align*}
\]

\[\bullet \ f = gr\] for some \(g \in XFUN_{w,v}\) and \(r \in XTERM^w\). In this case we have that \((gr) \sigma_U \cup \sigma_{U,\perp} \equiv (g \sigma_U \cup \sigma_{U,\perp})(r \sigma_U \cup \sigma_{U,\perp})\). Like in the preceding case we may apply the induction hypotheses:

\[
\begin{align*}
\text{DOM}(g^{\sigma_U^T D_U^T}) &= \text{DOM}((g \sigma_U \cup \sigma_{U,\perp})^T) \cap |ID_U^T|^w, \\
g^{\sigma_U^T D_U^T} x &= (g \sigma_U \cup \sigma_{U,\perp})^T x, \\
r^{\sigma_U^T D_U^T} &= (r \sigma_U \cup \sigma_{U,\perp})^T.
\end{align*}
\]

From the last equation we derive that \(r_i^{\sigma_U^T D_U^T} \neq t^T\) for all components \(r_i\) of \(r\) and all \(t \in T_U\), such that \(r_i, t \in XTERM_v\) for some \(v\). Using the equations above, we see that this implies that \(r^{\sigma_U^T D_U^T} \in \text{DOM}(g^{\sigma_U^T D_U^T})\) if and only if \((r \sigma_U \cup \sigma_{U,\perp})^T \in \text{DOM}((g \sigma_U \cup \sigma_{U,\perp})^T)\).

Now, if \(r^{\sigma_U^T D_U^T} \in \text{DOM}(g^{\sigma_U^T D_U^T})\), then by our observation above \((r \sigma_U \cup \sigma_{U,\perp})^T \in \text{DOM}((g \sigma_U \cup \sigma_{U,\perp})^T)\). So, by using Definition 4.1.11, we may infer that:

\[
\begin{align*}
(gr)^{\sigma_U^T D_U^T} &= g^{\sigma_U^T D_U^T} r^{\sigma_U^T D_U^T} \\
 &= (g \sigma_U \cup \sigma_{U,\perp})^T (r \sigma_U \cup \sigma_{U,\perp})^T \\
 &= (gr \sigma_U \cup \sigma_{U,\perp})^T.
\end{align*}
\]

If, on the other hand, \(r^{\sigma_U^T D_U^T} \notin \text{DOM}(g^{\sigma_U^T D_U^T})\), we also have \((r \sigma_U \cup \sigma_{U,\perp})^T \notin \text{DOM}((g \sigma_U \cup \sigma_{U,\perp})^T)\). Again using Definition 4.1.11, this yields \((gr)^{\sigma_U^T D_U^T} = \infty = (gr \sigma_U \cup \sigma_{U,\perp})^T\).

This completes the proof of the first part of the theorem.

Regarding the second part of the theorem, assume that \(I \models \text{Crem}(\varphi, T_U)\) holds. We have the following cases:

\[\bullet \ \varphi \equiv \text{tt} \text{ or } \varphi \equiv \text{ff}. \text{ Trivial}\]

\[\bullet \ \varphi \equiv \downarrow s. \text{ This case follows from the first part of this theorem, Definition 4.1.14, and the fact that } \left(\downarrow s \right) \sigma_U \cup \sigma_{U,\perp} \equiv \downarrow (s \sigma_U \cup \sigma_{U,\perp}).\]
6.1. The proof system

• $\varphi \equiv (r = s)$. Like the preceding case.

• $\varphi \equiv P r$ for some $P \in \text{Pred}_{w}$ and $r \in \text{Xterm}^{w}$. Now, we have that

\[(Pr) \sigma_{U} \cup \sigma_{U,\perp} = P(r \sigma_{U} \cup \sigma_{U,\perp}).\]

Applying the first part of the theorem to the components of $r$ yields $r \sigma_{U} \cup \sigma_{U,\perp} = t \sigma_{U} \cup \sigma_{U,\perp}$. Furthermore, $r_{t} \sigma_{U} \cup \sigma_{U,\perp} = t \sigma_{U} \cup \sigma_{U,\perp}$ for all components $r_{t}$ of $r$ and all $t \in T_{U}$, such that $r_{t} \sigma_{U} \cup \sigma_{U,\perp} = t \sigma_{U} \cup \sigma_{U,\perp}$ for some $v$. Since $P \sigma_{U} \cup \sigma_{U,\perp} = P \sigma_{U} \cup \sigma_{U,\perp}$, we may deduce that $r \sigma_{U} \cup \sigma_{U,\perp} = t \sigma_{U} \cup \sigma_{U,\perp}$ if and only if $(r \sigma_{U} \cup \sigma_{U,\perp})^{\perp} = (t \sigma_{U} \cup \sigma_{U,\perp})^{\perp}$, which settles this case.

• $\varphi \equiv \neg \psi$. From the fact that $(\neg \psi) \sigma_{U} \cup \sigma_{U,\perp} = \neg \psi \sigma_{U} \cup \sigma_{U,\perp}$, and from the induction hypothesis, which is supposed to hold for $\psi$, we infer:

\[I \sigma_{U}^{\perp} \models \neg \psi \quad \iff \quad \neg \sigma_{U}^{\perp} \models \psi \]

\[\quad \iff \quad I \models \neg \psi \sigma_{U} \cup \sigma_{U,\perp} \]

\[\quad \iff \quad I \models \neg \psi \sigma_{U} \cup \sigma_{U,\perp} \]

\[\quad \iff \quad I \models (\neg \psi) \sigma_{U} \cup \sigma_{U,\perp} \]

• $\varphi \equiv (\psi \ast \chi)$. Analogously as the preceding case.

• $\varphi \equiv \forall x^{v} \psi$. By Corollary 4.2.18 we may assume that $x^{v} \notin \text{FV}(t)$ for all $t \in T_{U}$, that $\sigma_{U}$ does not affect $x^{v}$, and that $x^{v} \notin \text{FV}(\sigma_{U})$. We will apply the induction hypothesis to $\psi$ with respect to the $\Sigma$-interpretation $I[x^{v} \mapsto d]$ where $d \neq t^{\perp}$ for all $t \in T_{U}$. This means we have to check for $I[x^{v} \mapsto d] \models \text{Crem}(\psi, T_{U})$. Using Theorem 4.1.17 and the fact that $d \neq t^{\perp}$ for all $t \in T_{U}$, it is easy to see that this condition is indeed fulfilled. The derivation runs as follows:

\[I \sigma_{U}^{\perp} \models \forall x^{v} \psi \]

\[\quad \iff \quad \forall d \in \sigma_{U}^{\perp} \models I[x^{v} \mapsto d] \sigma_{U}^{\perp} \models \psi \]

\[\quad \iff \quad \forall d \in \sigma_{U}^{\perp} \models I[x^{v} \mapsto d] \sigma_{U}^{\perp} \models \psi \]

\[\quad \iff \quad \forall d \in \sigma_{U}^{\perp} \models I[x^{v} \mapsto d] \sigma_{U}^{\perp} \models \psi \]

\[\quad \iff \quad \forall d \in \sigma_{U}^{\perp} \models I[x^{v} \mapsto d] \sigma_{U}^{\perp} \models \psi \]

\[\quad \iff \quad I \models \forall x^{v} \psi (\bigwedge_{t \in \text{TERM}_{U}} x^{v} \neq t \rightarrow \psi \sigma_{U} \cup \sigma_{U,\perp}^{\perp}) \]

\[\quad \iff \quad I \models (\forall x^{v} \psi) \sigma_{U} \cup \sigma_{U,\perp}^{\perp} \]

• $\varphi \equiv \exists x^{v} \psi$. Like the preceding case.

This completes the proof of Theorem 6.1.12. ■
The axiomatic semantics for EA we are about to define in this section, makes it possible to reason about the (partial and total) correctness of EAs. Reasoning is done in E-logic. Since it is useful to have auxiliary functions or predicates, the signature \( \Sigma_E \) of the E-logic to be used will often be richer than the signature \( \Sigma_{EA} \) of the EA considered. For instance, in Example 5.1.18 we needed the auxiliary functions \textit{string}, \textit{length} and \textit{at}. In these cases \( \Sigma_{EA} \) has to be embedded in \( \Sigma_E \).

But this is not the only condition that has to be satisfied. The standard parts of interpretations for \( \Sigma_E \) and \( \Sigma_{EA} \) have to be compatible as well. Let \( \Sigma_{EA} \) and \( \Sigma_E \) be the standard parts of \( \Sigma_{EA} \) and \( \Sigma_E \), respectively, let \( \mathcal{C} \) be the standard algebra for \( \Sigma_{EA} \), and let \( \mathcal{B} \) be the standard structure for \( \Sigma_E \). Then, of course, we want that all static sort symbols, function symbols and constant symbols of \( \Sigma_{EA} \) have the same interpretations in \( \mathcal{B} \) and \( \mathcal{C} \). Put differently, it has to be the case that \( \Sigma_{EA} \) is embedded in \( \Sigma_E \), and that \( \mathcal{B} \upharpoonright \Sigma_{EA} = \mathcal{C} \).

The last condition that has to be satisfied, is that dynamic sort symbols, function symbols and constant symbols of \( \Sigma_{EA} \) may not be interpreted by elements of \( \Sigma_E \). If all these conditions are fulfilled we say that \( \Sigma_{EA} \) is adequately embedded in \( \Sigma_E \) with respect to \( \mathcal{B} \).

\[ \text{6.1.13 Definition} \quad \text{Let} \quad \Sigma_{EA} \quad \text{be a signature for E-logic with standard part} \quad \Sigma_{EA}^s, \]  
\[ \text{and let} \quad \mathcal{B} \quad \text{be the standard structure for} \quad \Sigma_{EA}^s. \quad \text{Likewise, let} \quad \Sigma_E \quad \text{be a signature for} \quad \text{EA with standard part} \quad \Sigma_E^s. \]  
\[ \text{Then} \quad \Sigma_{EA} \quad \text{is said to be adequately embedded in} \quad \Sigma_E \quad \text{w.r.t.} \quad \mathcal{B} \quad \text{if the following three conditions hold:} \]

1. \( \Sigma_{EA} \subseteq \Sigma_E \), and \( \Sigma_{EA}^s \subseteq \Sigma_E^s \);
2. \( \mathcal{B} \upharpoonright \Sigma_{EA}^s \) is the standard algebra for \( \Sigma_{EA} \);
3. \( \text{DSORT}(\Sigma_{EA}) \cap \text{SORT}(\Sigma_E^s) = \emptyset \), and \( \text{DSYM}(\Sigma_{EA}) \cap \text{SYM}(\Sigma_E^s) = \emptyset \).

Due to the fact that signatures for EA differentiate between static and dynamic sorts and symbols, it is easy to define when a signature for EA is adequately embedded in another one. The notion of being adequately embedded for pairs of signatures for E-logic can be defined along the lines of the previous definition and will be left to the reader.

\[ \text{6.1.14 Definition} \quad \text{Let} \quad \Sigma_1 \quad \text{and} \quad \Sigma_2 \quad \text{be signatures for EA, with standard parts} \quad \Sigma_1^s \quad \text{and} \quad \Sigma_2^s, \quad \text{respectively. Moreover, let} \quad \mathcal{B} \quad \text{be the standard algebra for} \quad \Sigma_1^s. \]  
\[ \text{Then} \quad \Sigma_2 \quad \text{is said to be adequately embedded in} \quad \Sigma_1 \quad \text{w.r.t.} \quad \mathcal{B} \quad \text{if the following two conditions hold:} \]

1. \( \Sigma_2 \subseteq \Sigma_1 \); and
2. \( \mathcal{B} \upharpoonright \Sigma_2^s \) is the standard algebra for \( \Sigma_2^s \).

In proving the correctness of an EA it is often useful to prove the correctness of another EA which uses auxiliary sorts, functions or constants. These auxiliary sorts, functions and constants may, of course, neither influence the control flow, i.e. the order in which the rules of the EA considered are executed, nor
the values of the dynamic symbols of the original EA. The best way to visualize
the execution of the 'richer' EA is that it does some additional book keeping
on a separate piece of paper. The idea is to throw this piece of paper away at
the end when the correctness of that EA has been proved, thus yielding the
correctness of the original EA.

For example, to prove the correctness of the EA of Example 5.1.18 it will
turn out to be very practical to record some additional information for each
node of the stack. In this case, we will record its serial number (called index).
This makes it possible to directly relate the position of a character in a string
to the index of a node in the stack.

The idea of auxiliary symbols is not new. In [Cli73] a correctness proof
is given using so-called mythical variables. Updates of these variables in a
program are put between quotes, meaning that these can be ignored during
runtime. They are only used in proving the correctness. The authors of [OG76]
introduce a proof rule for auxiliary variables in this way formalizing the idea of
[Cli73]. We will do so as well in the context of EAs. Firstly, we will define the
concept of an auxiliary update, the concept of a rule using auxiliary updates,
and the concept of an EA using auxiliary updates.

6.1.15 CONVENTION For the remainder of this chapter, we will assume that
$\Sigma_E$ is a signature for $E$-logic with standard part $\Sigma_E^s$, and that $\mathcal{B}$ is the standard
structure for $\Sigma_E^s$. Moreover, we will assume that $\Sigma_{EA}$ is a signature for EA with
standard part $\Sigma_{EA}^s$, and that $\Sigma_{EA}^{as}$ is an auxiliary signature with respect to $\Sigma_{EA}$,
such that $\Sigma_{EA} \subseteq \Sigma_{EA}^{as} \subseteq \Sigma_E$. Furthermore, we assume that all these embeddings
are adequate with respect to $\mathcal{B}$.

6.1.16 DEFINITION An update $U \in \text{UPD}(\Sigma_{EA}^{as})$ is called an auxiliary update,
if $U$ has one of the following formats:

- $c := t$, where $c \in \text{DSYM}(\Sigma_{EA}^{as}) \setminus \text{DSYM}(\Sigma_{EA})$;
- $fs := t$, where $f \in \text{DSYM}(\Sigma_{EA}^{as}) \setminus \text{DSYM}(\Sigma_{EA})$;
- $\text{rem } t : u$, where $u \in \text{DSORT}(\Sigma_{EA}^{as}) \setminus \text{DSORT}(\Sigma_{EA})$;
- $\text{new } x : u$ with $\mathcal{U}$, where $u \in \text{DSORT}(\Sigma_{EA}^{as}) \setminus \text{DSORT}(\Sigma_{EA})$, and where all elements of $\mathcal{U}$ are auxiliary updates.

6.1.17 DEFINITION Let $R^0 \in \text{RULE}(\Sigma_{EA})$ and let $R \in \text{RULE}(\Sigma_{EA}^{as})$. Then $R$
is said to contain auxiliary updates with respect to $R^0$, $\Sigma_{EA}$ and $\Sigma_{EA}^{as}$, if the
following conditions hold:

1. $R? \equiv R^0?$ (implying that $R? \in \text{COND}(\Sigma_{EA})$);
2. $R^0!$ can be obtained from $R!$ by deleting all auxiliary updates occurring
   (nested or not nested) in $R!$;
3. $\text{Cons } (R^0!, \mathcal{I})$ implies $\text{Cons } (R!, \mathcal{I})$ for any $\Sigma_{EA}^{as}$-interpretation $\mathcal{I}$. 
The last, semantic condition in the preceding definition (in combination with the first, syntactic condition) assures that the auxiliary updates in \( R \) will not disturb the runtime behaviour of \( R^0 \).

**6.1.18 Definition** Let \( R^0 = \{ R^0_1, \ldots, R^0_n \} \subseteq \text{fin} \ \text{Rule}(\Sigma_{EA}) \) and let \( R = \{ R_1, \ldots, R_n \} \subseteq \text{fin} \ \text{Rule}(\Sigma_{EA}^{\text{aux}}) \). Then \( R \) is said to contain auxiliary updates with respect to \( R^0, \Sigma_{EA} \) and \( \Sigma_{EA}^{\text{aux}} \), if all \( R_i \) contain auxiliary updates with respect to \( R^0_i, \Sigma_{EA} \) and \( \Sigma_{EA}^{\text{aux}} \) (1 \( \leq i \leq n \)).

The notion of a restricted signature \( \Sigma_{E}^- \) for E-logic to be introduced in the next definition, will be used in the derivation rule for auxiliary updates. This kind of signatures does not contain auxiliary symbols.

**6.1.19 Definition** Let \( \Sigma_{E}^- \) be a signature for E-logic. Then \( \Sigma_{E}^- \) is called a **signature restricted** with respect to \( \Sigma_{EA} \), \( \Sigma_{EA}^{\text{aux}} \) and \( \Sigma_{E} \), if the following conditions are met:

1. \( \Sigma_{EA} \subseteq \Sigma_{E}^- \subseteq \Sigma_{E} \);
2. \( \text{Sym}(\Sigma_{E}^-) \cap (\text{Dsym}(\Sigma_{EA}^{\text{aux}}) \setminus \text{Dsym}(\Sigma_{EA})) = \emptyset \);
3. \( \text{Sort}(\Sigma_{E}^-) \cap (\text{Dsort}(\Sigma_{EA}^{\text{aux}}) \setminus \text{Dsort}(\Sigma_{EA})) = \emptyset \).

**6.1.20 Convention** For the remainder of this chapter, we will assume that \( \Sigma_{E}^- \) denotes some signature restricted w.r.t. \( \Sigma_{EA}, \Sigma_{EA}^{\text{aux}} \) and \( \Sigma_{E} \).

The **extend condition** \( \text{Cnew}(\varphi, x) \), to be defined in the next definition, is used in the formalization of the derivation rule for extension updates. It resembles the remove condition (see Definition 6.1.8). But, in fact, it is a stronger condition: it also expresses that no function mentioned in \( \varphi \) is defined on the ‘new’ element denoted by \( x \).

**6.1.21 Definition** Let \( x^u \in \text{VAR}_u \) and \( \varphi \in \text{FORM} \). Define the following sets:

- \( F_{\varphi} \) is the set of all function symbols \( f \) occurring in \( \varphi \) such that \( f \in \text{Fun}_{w,u} \) for some \( w \in \text{Sort}^+ \);
- \( F_{\varphi}' \) is the set of all function symbols \( f \) occurring in \( \varphi \) such that \( f \in \text{Fun}_{w,v} \) for some \( v \in \text{Sort} \) and \( w = u_1 \ldots u_n \in \text{Sort}^+ \), where \( u_i = u \) for some \( i \) (1 \( \leq i \leq n \));
- \( C_{\varphi} \) is the set of all constant symbols \( c \) occurring in \( \varphi \) such that \( c \in \text{Con}_u \);
- \( X_{\varphi} \) is the set of all variables \( y^u \in \text{VAR}_u \cap \text{Fv}(\varphi) \).

Then the **extend condition** \( \text{Cnew}(\varphi, x^u) \) is defined by:

\[
\downarrow x^u \land \bigwedge_{f \in F_{\varphi}} \forall x(f x \neq x^u) \land \bigwedge_{c \in C_{\varphi} \cup X_{\varphi}} c \neq x^u \\
\bigwedge_{f \in F_{\varphi}'} \forall (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \neg \downarrow f(x_1, \ldots, x_{i-1}, x^u, x_{i+1}, \ldots, x_n),
\]

\[
1 \leq i \leq n.
\]
where all variables $x$ are fresh, i.e. there is no component $x_i$ of $x$ such that $x_i \equiv x^u$, and where it is supposed that $f(x_1, \ldots, x_{i-1}, x^u, x_{i+1}, \ldots, x_n)$ is well-formed, i.e. $(x_1, \ldots, x_{i-1}, x^u, x_{i+1}, \ldots, x_n) \in \text{VAR}^w$ whenever $f \in \text{FUN}_{w,v}$.

6.1.22 Proposition Let $\varphi \in \text{FORM}$ and $x^u \in \text{VAR}_u$. Then:

$$\text{Cnew}(\varphi, x^u) \vdash \text{Crem}(\varphi, \{x^u\}).$$

Proof This directly follows from Definitions 6.1.21 and 6.1.8. 

6.1.23 Proposition Let $\mathcal{I}$ be a $\Sigma$-interpretation, $\varphi \in \text{FORM}$, and $x \in \text{VAR}_u$. Then:

$$\text{TE}_d^u[x \mapsto d] \models \text{Cnew}(\varphi, x).$$

Proof This is a consequence of Definition 4.2.4. 

Now we have all necessary machinery to define the axiomatic semantics for partial correctness of evolving algebras. We will firstly define the notion of a Hoare-formula or correctness formula.

A correctness formula is a triplet consisting of a precondition $\varphi$, (a part of) an EA, say $\Theta$, and a postcondition $\psi$. Such a triplet will be rendered as:

$$\{\varphi\} \Theta \{\psi\}.$$

The intuitive reading is as follows. If $I$ is a state in which $\varphi$ holds, and if a computation of $\Theta$ starting in $I$ ends in state $J$, then $\psi$ holds in $J$. Note, that if $\{\varphi\} \Theta \{\psi\}$ holds, we do not know anything about the termination of $\Theta$.

6.1.24 Definition Let $\varphi, \psi \in \text{FORM}(\Sigma_E)$. Moreover, let $\Theta \subseteq_{\text{fin}} \text{UPD}(\Sigma^{\text{eva}}_E)$, or $\Theta \in \text{Rule}(\Sigma^{\text{eva}}_E)$, or $\Theta \subseteq_{\text{fin}} \text{Rule}(\Sigma^{\text{eva}}_E)$. Then $\{\varphi\} \Theta \{\psi\}$ is called a correctness formula. The formula $\varphi$ is called the precondition, whereas $\psi$ is called the postcondition of the correctness formula.

The correctness formulae we actually want to prove have the format $\{\varphi\} \mathcal{R} \{\psi\}$, where $\varphi, \psi \in \text{FORM}(\Sigma_E^\rightarrow)$ and $\mathcal{R} \subseteq_{\text{fin}} \text{UPD}(\Sigma_E^\rightarrow)$. So, we want that the precondition $\varphi$ and the postcondition $\psi$ do not contain auxiliary symbols. But, when auxiliary symbols have been used in the proof, we will firstly have a correctness formula $\{\varphi \land \psi\} \mathcal{R} \{\chi\}$, where $\mathcal{R}$ does contain auxiliary updates, where $\varphi, \chi \in \text{FORM}(\Sigma_E^\rightarrow)$ and $\psi \in \text{FORM}(\Sigma_E)$. The formula $\psi$ will contain initializations of the auxiliary symbols. Of course, in the final correctness formula we want to have gotten rid of these initializations.

When adding an initialization $\psi$ of the auxiliary symbols to a precondition $\varphi$ we have to be careful. It has to be compatible with the precondition $\varphi$ in the following sense. The class of initial states for the EA not containing the auxiliary updates may not be restricted by $\psi$. This property is defined in the following.
6.1.25 Definition Let $\varphi \in \text{Form}(\Sigma_E^E)$ and $\psi \in \text{Form}(\Sigma_E)$. Then $\psi$ is called compatible with respect to $\varphi$, if for all $\Sigma_E^E$-interpretations $I$ such that $I \models \varphi$, there exists a $\Sigma_E$-interpretation $J$ such that $J \models \varphi \land \psi$ and $J \upharpoonright \Sigma_E^E = I$.

6.1.26 Definition Let $\pi, \varphi, \psi, \chi \in \text{Form}(\Sigma_E)$, and let $\mathcal{U} \subseteq \text{fin} \ \text{UPD}(\Sigma_{EA}^*)$. The axiomatic semantics $\mathcal{P}$ for partial correctness of evolving algebras is defined by the following axiom scheme and derivation rules.

Axiom for simple updates

\[
\text{LocRem} : \ \{C(\mathcal{U}) \land \text{Crem}(\varphi, T_{\mathcal{U}}) \land \overline{\varphi_{\sigma_{\mathcal{U}}} \cup \sigma_{\mathcal{U}} \upharpoonright T_{\mathcal{U}}}\} \ \mathcal{U} \ \{\varphi\},
\]

where $\mathcal{U}$ only contains simple updates.

Rule for extension updates

\[
\text{Ext} : \ \begin{cases} C_{\text{new}}(\varphi, y) \land \overline{\varphi^y} \ \mathcal{U}[x := y] \ \{\psi\} \\ \{\varphi\} \ \text{new} \ x : u \text{ with } \mathcal{U} \ \{\psi\} \end{cases},
\]

where $y \in \text{VAR}_u$ is such that $y \notin \text{FV}(\{\varphi, \psi\}) \cup (\text{FV}(\mathcal{U}) \setminus \{x\})$.

Structural rule

\[
\text{Norm} : \ \begin{cases} \{\varphi\} \ \text{Norm}(\mathcal{U}) \ \{\psi\} \\ \{\varphi\} \ \mathcal{U} \ \{\psi\} \end{cases},
\]

where $\text{Norm}(\mathcal{U})$ denotes a possible result of the operation $\text{Norm}(\mathcal{U})$ when applied to $\mathcal{U}$.

Rule for EA-rules

\[
\text{If} : \ \begin{cases} \{\varphi \land R?\} \ \mathcal{R}! \ \{\psi\} \\ \{\varphi\} \ \mathcal{R} \ \{\psi\} \end{cases},
\]

where $\mathcal{R} \in \text{RULE}(\Sigma_{EA}^*)$ and $R? \in \text{COND}(\Sigma_{EA})$.

Rule for EAs

\[
\text{Ealg} : \ \begin{cases} \{\varphi\} \ R_1 \ \{\varphi\} \ \ldots \ \{\varphi\} \ R_n \ \{\varphi\} \\ \{\varphi\} \ \mathcal{R} \ \{\varphi \land \lnot R_1? \land \ldots \land \lnot R_n?\} \end{cases},
\]

where $\mathcal{R} = \{R_1, \ldots, R_n\} \subseteq \text{fin} \ \text{RULE}(\Sigma_{EA}^*)$, and $\mathcal{R}$ is consistent relative to $\varphi$. 
6.1. The proof system

Logical rule

\[
\begin{align*}
\text{Conseq:} & \quad \varphi \vdash_{\Theta} \pi \quad \{\pi\} \Theta \{\chi\} \quad \chi \vdash_{\Theta} \psi, \\
& \quad \{\varphi\} \Theta \{\psi\},
\end{align*}
\]

where $\Theta \subseteq_{\text{fin}} \text{UPD}(\Sigma_{EA}^{\text{ext}})$, or $\Theta \in \text{RULE}(\Sigma_{EA}^{\text{ext}})$, or $\Theta \subseteq_{\text{fin}} \text{RULE}(\Sigma_{EA}^{\text{ext}})$.

Rule for auxiliary updates

\[
\begin{align*}
\text{Aux:} & \quad \{\theta \land \varphi\} \mathcal{R} \{\xi\} \\
& \quad \{\theta\} \mathcal{R}^0 \{\xi\},
\end{align*}
\]

where $\mathcal{R}$ contains auxiliary updates with respect to $\mathcal{R}^0$, $\Sigma_{EA}$ and $\Sigma_{EA}^{\text{ext}}$, where $\theta, \xi \in \text{FORM}(\Sigma_{E}^{-})$, and where $\varphi$ is compatible with respect to $\theta$.

6.1.27 Definition A derivation or proof of a correctness formula $\Phi$ is a series $\Phi_1, \ldots, \Phi_n \equiv \Phi$, such that for any $i$ ($1 \leq i \leq n$):

- $\Phi_i$ is an instantiation of the axiom scheme $\text{LocRem}$, or
- $\Phi_i$ is a theorem of the form $\varphi \vdash_{\Theta} \psi$, for some $\varphi, \psi \in \text{FORM}(\Sigma_{E})$, or
- $\Phi_i$ is the result of applying a derivation rule on theorems or correctness formulae $\Phi_j$ preceding $\Phi_i$, i.e. for $1 \leq j < i$.

If such a derivation exists, $\Phi$ is called derivable or provable, notation $\vdash_{\mathcal{P}} \Phi$. Moreover, $n$ is called the length of the derivation.

In order to define the notion of validity for correctness formulae, we have a small problem. In a correctness formula $\{\varphi\} \mathcal{R} \{\psi\}$, where $\varphi \in \text{FORM}(\Sigma_{E})$ and $\mathcal{R} \subseteq_{\text{fin}} \text{UPD}(\Sigma_{EA}^{\text{ext}})$, we have to deal with $\Sigma_{E}$-interpretations (for the precondition and the postcondition), and with $\Sigma_{EA}^{\text{ext}}$-interpretations (for $\mathcal{R}$).

The problem can be solved if we let EAs operate on $\Sigma_{E}$-interpretations. What then are the effects of an EA upon a $\Sigma_{E}$-interpretation? Well, if we look at the rules of the operational semantics, and the adequacy theorems and their proofs (see Chapter 5), we see that these rules and theorems can also be meaningfully interpreted taking $\Sigma_{E}$-interpretations in stead of $\Sigma_{EA}^{\text{ext}}$-interpretations. The proofs of the theorems mentioned will be still valid, since nowhere a reference is made to the fact that we are dealing with $\Sigma_{EA}^{\text{ext}}$-interpretations.

We would also like to know the relation of the effect of a run of an EA upon a $\Sigma_{EA}^{\text{ext}}$-interpretation, and its corresponding effect upon a $\Sigma_{E}$ interpretation. This relation is very simple: the changes made in the $\Sigma_{EA}^{\text{ext}}$-interpretation are also made in the $\Sigma_{E}$-interpretation. The converse, however, need not to be true, since there can be side effects upon the interpretation of symbols of $\Sigma_{E}$ which are not in $\Sigma_{EA}$.

The relation is made precise in the following theorems, which are stated for $\Sigma_{EA}$. Since $\Sigma_{EA}^{\text{ext}}$ is a signature for EA such that $\Sigma_{EA}^{\text{ext}} \subseteq \Sigma_{E}$, the theorem also holds for $\Sigma_{EA}^{\text{ext}}$ with respect to $\Sigma_{E}$. Likewise for $\Sigma_{EA}$ with respect to $\Sigma_{E}^{-}$.
6.1.28 Theorem i. Let $\mathcal{I}$ and $\mathcal{J}$ be $\Sigma_{EA}$-interpretations. Moreover, let $\mathcal{R} \subseteq_{fin} \text{RULE}(\Sigma_{EA})$, and $\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{J}$. Then $\mathcal{I} \upharpoonright \Sigma_{EA} \xrightarrow{\mathcal{R}} \mathcal{J} \upharpoonright \Sigma_{EA}$.

ii. Let $\mathcal{I}_1$ and $\mathcal{J}_1$ be $\Sigma_{EA}$-interpretations, and let $\mathcal{I}_2$ be a $\Sigma_{EA}$-interpretation such that $\mathcal{I}_2 \upharpoonright \Sigma_{EA} = \mathcal{I}_1$. Moreover, let $\mathcal{R} \subseteq_{fin} \text{RULE}(\Sigma_{EA})$, and let $\mathcal{I}_1 \xrightarrow{\mathcal{R}} \mathcal{J}_1$. Then, there exists a $\Sigma_{EA}$-interpretation $\mathcal{J}_2$ such that $\mathcal{I}_2 \xrightarrow{\mathcal{R}} \mathcal{J}_2$ and $\mathcal{J}_2 \upharpoonright \Sigma_{EA}$ and $\mathcal{J}_1$ are isomorphic.

Proof

Part i Assume that $\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{J}$. Then there exists a rule $R \in \mathcal{R}$, such that $\mathcal{I} \models R$? and Cons $(R!, \mathcal{I})$, and:

$$\langle R! \mid \mathcal{I} \rangle \xrightarrow{\mathcal{T}} \langle \emptyset \mid \mathcal{J} \rangle.$$  

Since all symbols occurring in $R!$ and $R!$ belong to $\Sigma_{EA}$, we infer by applying Theorem 4.1.17 that $\mathcal{I} \upharpoonright \Sigma_{EA} \models R$ and Cons $(R!, \mathcal{I} \upharpoonright \Sigma_{EA})$. This means that $R!$ can be executed upon $\mathcal{I} \upharpoonright \Sigma_{EA}$ whenever it can be executed upon $\mathcal{I}$. We will prove by induction on the complexity $\text{COMP} (R!)$, that:

$$\langle R! \mid \mathcal{I} \upharpoonright \Sigma_{EA} \rangle \xrightarrow{\mathcal{T} \upharpoonright \Sigma_{EA}} \langle \emptyset \mid \mathcal{J} \upharpoonright \Sigma_{EA} \rangle,$$

from which this part of the theorem follows.

For the base case, assume that $\text{COMP} (R!) = 0$. From Lemmas 5.3.5 and 5.3.6, it follows that $\mathcal{J} = \mathcal{IO}_1 \ldots \mathcal{O}_m$ for some operators $\mathcal{O}_j$ ($1 \leq j \leq m$), where all operators $\mathcal{O}_j$ correspond to an update $U \in \text{UPD}(\Sigma_{EA})$. Each operator $\mathcal{O}_j$ is the result of evaluating its corresponding update $U$ with respect to evaluation state $\mathcal{I}$. Now, let $\mathcal{O}'_j$ be the result of evaluating $U$ with respect to $\mathcal{I} \upharpoonright \Sigma_{EA}$. Then, by applying Theorem 4.1.17 again, we infer that $\mathcal{O}'_j = \mathcal{O}_j$, since $U \in \text{UPD}(\Sigma_{EA})$. Moreover, $(\mathcal{K} \upharpoonright \Sigma_{EA})\mathcal{O}'_j = (\mathcal{K}\mathcal{O}_j) \upharpoonright \Sigma_{EA}$ for all $\Sigma_{EA}$-interpretations $\mathcal{K}$ and all $\mathcal{O}_j$ (and $\mathcal{O}'_j$). Since $\mathcal{I} \upharpoonright \Sigma_{EA} | u = | \mathcal{I} | u$ for all $u \in \text{SORT}(\Sigma_{EA})$, and $f(\mathcal{T} \upharpoonright \Sigma_{EA}) = f\mathcal{T}$ for all symbols $f \in \text{SYM}(\Sigma_{EA})$, it now follows by induction on $m$ that:

$$| (\mathcal{I} \upharpoonright \Sigma_{EA}) \mathcal{O}_1' \ldots \mathcal{O}_m' | u = | (\mathcal{I} \mathcal{O}_1 \ldots \mathcal{O}_m) \upharpoonright \Sigma_{EA} | u$$

$$= | \mathcal{J} \upharpoonright \Sigma_{EA} | u,$$

for all $u \in \text{SORT}(\Sigma_{EA})$, and:

$$f(\mathcal{T} \upharpoonright \Sigma_{EA}) \mathcal{O}_1' \ldots \mathcal{O}_m' = f(\mathcal{I} \mathcal{O}_1 \ldots \mathcal{O}_m) \upharpoonright \Sigma_{EA}$$

$$= f\mathcal{J} \upharpoonright \Sigma_{EA},$$

for all $f \in \text{SYM}(\Sigma_{EA})$. But this means that:

$$(\mathcal{I} \upharpoonright \Sigma_{EA}) \mathcal{O}_1' \ldots \mathcal{O}_m' = \mathcal{J} \upharpoonright \Sigma_{EA},$$

which implies that:

$$\langle R! \mid \mathcal{I} \upharpoonright \Sigma_{EA} \rangle \xrightarrow{\mathcal{T} \upharpoonright \Sigma_{EA}} \langle \emptyset \mid \mathcal{J} \upharpoonright \Sigma_{EA} \rangle.$$
For the induction step, suppose that \( \text{COMP} (R!) = n + 1 \). Without loss of generality, assume that \( R! \) is in normal form, say \( R! \equiv \text{new } x : u \) with \( \mathcal{U} \), where \( \text{COMP} (\mathcal{U}) = n \). Since:
\[
\langle R! \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle,
\]
we also must have:
\[
\langle \mathcal{U} \mid \mathcal{I} \mathcal{E}_d^u [x \mapsto d] \rangle \xrightarrow{\mathcal{I} \mathcal{E}_d^u [x \mapsto d]} \langle \emptyset \mid \mathcal{K} \rangle,
\]
for some \( \Sigma_{\mathcal{E}} \)-interpretation \( \mathcal{K} \) such that \( \mathcal{J} = \mathcal{K}[x \mapsto d'] \) for some suitable \( d' \).
From this fact we infer that \( \mathcal{K} = \mathcal{J}[x \mapsto x^K] \). By the induction hypothesis, we have:
\[
\langle \mathcal{U} \mid (\mathcal{I} \mathcal{E}_d^u [x \mapsto d]) \mid \Sigma_{\mathcal{E}} \rangle \xrightarrow{(\mathcal{I} \mathcal{E}_d^u [x \mapsto d]) \mid \Sigma_{\mathcal{E}}} \langle \emptyset \mid (\mathcal{J}[x \mapsto x^K]) \mid \Sigma_{\mathcal{E}} \rangle.
\]
As \( u \in \text{SORT}(\Sigma_{\mathcal{E}}) \), this is equivalent to:
\[
\langle \mathcal{U} \mid (\mathcal{I} \mid \Sigma_{\mathcal{E}}) \mathcal{E}_d^u [x \mapsto d] \rangle \xrightarrow{(\mathcal{I} \Sigma_{\mathcal{E}}) \mathcal{E}_d^u [x \mapsto d]} \langle \emptyset \mid (\mathcal{J} \mid \Sigma_{\mathcal{E}})[x \mapsto x^K] \rangle.
\]
Applying the rule for extension updates now yields:
\[
\langle R! \mid \mathcal{I} \mid \Sigma_{\mathcal{E}} \rangle \xrightarrow{\mathcal{I} \Sigma_{\mathcal{E}}} \langle \emptyset \mid (\mathcal{J} \mid \Sigma_{\mathcal{E}})[x \mapsto x^K][x \mapsto d''] \rangle
\]
\[
= \langle \emptyset \mid (\mathcal{J} \mid \Sigma_{\mathcal{E}})[x \mapsto x^K][x \mapsto d''] \mid \Sigma_{\mathcal{E}} \rangle
\]
\[
= \langle \emptyset \mid (\mathcal{K}[x \mapsto d'']) \mid \Sigma_{\mathcal{E}} \rangle.
\]
As \( |\mathcal{K}|_u = |\mathcal{K} \mid \Sigma_{\mathcal{E}}|_u \), it follows from the definition of \( d' \) and \( d'' \) (see Definition 5.2.33) that \( d'' = d' \). This enables us to rewrite the last transition as:
\[
\langle R! \mid \mathcal{I} \mid \Sigma_{\mathcal{E}} \rangle \xrightarrow{\mathcal{I} \Sigma_{\mathcal{E}}} \langle \emptyset \mid \mathcal{J} \mid \Sigma_{\mathcal{E}} \rangle,
\]
which concludes the proof of part i.

**Part ii** Suppose that \( \mathcal{I}_1 \xrightarrow{\mathcal{R}} \mathcal{J}_1 \) and \( \mathcal{I}_2 \mid \Sigma_{\mathcal{E}} = \mathcal{I}_1 \). This implies that for some rule \( R \in \mathcal{R} \), we have \( \mathcal{I}_1 \models R? \) and \( \text{Cons} (R!, \mathcal{I}_1) \), and:
\[
\langle R! \mid \mathcal{I}_1 \rangle \xrightarrow{\mathcal{I}_1} \langle \emptyset \mid \mathcal{J}_1 \rangle.
\]
From these facts we derive that \( \mathcal{I}_2 \models R? \) and \( \text{Cons} (R?, \mathcal{I}_2) \) by applying Theorem 4.1.17. So, by the Completeness Theorem 5.3.27, there exists a \( \Sigma_{\mathcal{E}} \)-interpretation \( \mathcal{J}_2 \) such that:
\[
\langle R! \mid \mathcal{I}_2 \rangle \xrightarrow{\mathcal{I}_2} \langle \emptyset \mid \mathcal{J}_2 \rangle.
\]
Applying the first part of this theorem to this transition gives us:
\[
\langle R! \mid \mathcal{I}_2 \mid \Sigma_{\mathcal{E}} \rangle \xrightarrow{\mathcal{I}_2 \mid \Sigma_{\mathcal{E}}} \langle \emptyset \mid \mathcal{J}_2 \mid \Sigma_{\mathcal{E}} \rangle.
\]
But this means that \( \mathcal{I}_2 \mid \Sigma_{\mathcal{E}} \xrightarrow{\mathcal{R}} \mathcal{J}_2 \mid \Sigma_{\mathcal{E}} \). Since \( \mathcal{I}_2 \mid \Sigma_{\mathcal{E}} = \mathcal{I}_1 \), we have by the Soundness Theorem 5.3.26 that \( \mathcal{J}_2 \mid \Sigma_{\mathcal{E}} \) and \( \mathcal{J}_1 \) are isomorphic. This completes the proof of Theorem 6.1.28.  ■
One might wonder why in the second part of the preceding theorem we do not have that $\mathcal{I}_2 | \Sigma_{EA} = \mathcal{J}_1$. Since $\Sigma_{EA} \subseteq \Sigma_E$, it might be the case that $\mathcal{I}_2$ possesses more universes than $\mathcal{I}_1$. So, is will not be always possible to start a transition from $\mathcal{I}_2$ using the same set $D$ of reserve elements as in a transition from $\mathcal{I}_1$, as it is required that $D$ is disjoint from all universes of $\mathcal{I}_2$. The fact that $D$ is disjoint from all universes of $\mathcal{I}_1$ does not guarantee that $D$ is disjoint from all universes of $\mathcal{I}_2$.

We will now generalize the theorem for the relation $\mathcal{R} \rightarrow$.

6.1.29 Theorem i. Let $\mathcal{I}$ be a $\Sigma_E$-interpretation. Furthermore, let $\mathcal{R} \subseteq_{\text{fin}} \text{Rule}(\Sigma_{EA})$, and let $\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{J}$, where $\mathcal{J}$ is a final state. Then $\mathcal{I} | \Sigma_{EA} \xrightarrow{\mathcal{R}} \mathcal{J} | \Sigma_{EA}$ and $\mathcal{J} | \Sigma_{EA}$ is a final state.

ii. Let $\mathcal{I}_1$ be a $\Sigma_{EA}$-interpretation, and let $\mathcal{I}_2$ be a $\Sigma_E$-interpretation such that $\mathcal{I}_2 | \Sigma_{EA} = \mathcal{I}_1$. Furthermore, let $\mathcal{R} \subseteq_{\text{fin}} \text{Rule}(\Sigma_{EA})$, and let $\mathcal{I}_1 \xrightarrow{\mathcal{R}} \mathcal{J}_1$, where $\mathcal{J}_1$ is a final state. Then, there exists a $\Sigma_E$-interpretation $\mathcal{J}_2$ such that $\mathcal{I}_2 \xrightarrow{\mathcal{R}} \mathcal{J}_2$ where $\mathcal{J}_2$ is a final state. Moreover, $\mathcal{J}_2$ isomorphic to $\mathcal{J}_1$.

Proof The proof of this theorem is by induction on the number of $\mathcal{R} \rightarrow$-transitions, using Theorem 6.1.28 and Theorem 4.1.17.

The previous theorems make it possible to give an adequate definition of the notion of validity (in the sense of partial correctness) of correctness formulae.

6.1.30 Definition Let $\Phi \equiv \{\varphi\} \Theta \{\psi\}$ be a correctness formula. Then $\Phi$ is valid in the sense of partial correctness, notation $\vdash_P \Phi$, if one of the following conditions holds:

- $\Theta \subseteq_{\text{fin}} \text{UPD}(\Sigma_{EA}^{\text{out}})$, and for all standard $\Sigma_E$-interpretations $\mathcal{I}$:

$$\mathcal{I} \models \Theta \varphi \land (\Theta | \mathcal{I}) \xrightarrow{\mathcal{I}} (\emptyset | \mathcal{J}) \Rightarrow \mathcal{J} \models \Theta \psi.$$  

- $\Theta \in \text{Rule}(\Sigma_{EA}^{\text{out}})$, and for all standard $\Sigma_E$-interpretations $\mathcal{I}$:

$$\mathcal{I} \models \Theta \varphi \land \mathcal{I} \xrightarrow{\Theta} \mathcal{J} \Rightarrow \mathcal{J} \models \Theta \psi.$$  

- $\Theta \subseteq_{\text{fin}} \text{Rule}(\Sigma_{EA}^{\text{out}})$, and for all standard $\Sigma_E$-interpretations $\mathcal{I}$:

$$\mathcal{I} \models \Theta \varphi \land \mathcal{I} \xrightarrow{\Theta} \mathcal{J} \land \mathcal{J} \text{ is a final state} \Rightarrow \mathcal{J} \models \Theta \psi.$$  

In this case, $\Theta$ is called partially correct with respect to precondition $\varphi$ and postcondition $\psi$.

We will prove that the system $\mathcal{P}$ is sound. However, the system is not complete, as will be shown by a counter example.
6.1.31 Theorem Soundness $\mathcal{P}$

For all correctness formulae $\Phi$:

$$\vdash_{\mathcal{P}} \Phi \Rightarrow \vdash_{\mathcal{P}} \Phi.$$

Proof. We will prove the theorem by induction on the length of the derivation, where we do not count the number of theorems $\varphi \models_{\mathcal{B}} \psi$. To begin with, note that the theorem trivially holds in case the length is 0, since that is not possible. So, assume that the length is $n + 1$, and that the theorem holds for length $n$. Moreover, let $\Phi$ be the correctness formula at the end of the derivation. Then, $\Phi$ can be the result of applying one the following axiom or rules:

- **Locrem.** In this case we have that $\Phi$ is the correctness formula:

$$\{ C(\mathcal{U}) \land \text{Crem} (\varphi, T_\mathcal{U}) \land \overline{\varphi \sigma \mathcal{U} \cup \sigma_{\mathcal{U}, \perp} T_\mathcal{U}} \} \cup \{ \varphi \},$$

where $\mathcal{U} \subseteq_{\text{fin}} \text{UPD}(\Sigma_{\mathcal{U}, \mathcal{A}})$ only contains simple updates. Now, suppose that $\mathcal{I}$ is a $\Sigma_\varphi$-interpretation, such that:

$$\mathcal{I} \models_{\mathcal{B}} C(\mathcal{U}) \land \text{Crem} (\varphi, T_\mathcal{U}) \land \overline{\varphi \sigma \mathcal{U} \cup \sigma_{\mathcal{U}, \perp} T_\mathcal{U}},$$

and:

$$\langle \mathcal{U} \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle.$$

We then have to show that $\mathcal{J} \models_{\mathcal{B}} \varphi$.

By Lemmas 5.3.5 and 5.3.6, we have that $\mathcal{J} = \mathcal{I} O_1 \ldots O_n$, where each operator $O_i$ is a local modification operator or remove operator. From the fact that $\mathcal{I} \models_{\mathcal{B}} C(\mathcal{U})$ it easily follows that:

$$\mathcal{J} = \mathcal{I} O_1 \ldots O_n = \mathcal{I} \sigma_{\mathcal{U}, \mathcal{D}} T_\mathcal{U}.$$

Since we also have $\mathcal{I} \models_{\mathcal{B}} \text{Crem} (\varphi, T_\mathcal{U}) \land \overline{\varphi \sigma \mathcal{U} \cup \sigma_{\mathcal{U}, \perp} T_\mathcal{U}}$, we can apply Theorem 6.1.12, which yields $\mathcal{J} \models_{\mathcal{B}} \varphi$.

- **Ext.** Now, $\Phi$ is the correctness formula:

$$\{ \varphi \} \text{ new } x : u \text{ with } \mathcal{U} \{ \psi \}.$$

Assume that:

$$\{ \text{Cnew} (\varphi, y) \land \overline{\varphi y} \} \mathcal{U}[x := y] \{ \psi \}, \quad (6.1)$$

where $y \in \text{VAR}_u$ is such that $y \notin \text{FV} (\{ \varphi, \psi \}) \cup (\text{FV} (\mathcal{U}) \setminus \{ x \})$. Moreover, assume that $\mathcal{I} \models_{\mathcal{B}} \varphi$ and:

$$\langle \text{new } x : u \text{ with } \mathcal{U} \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle, \quad (6.2)$$
for some $\Sigma_{\mathcal{E}}$-interpretation $\mathcal{J}$. We then have to show that $\mathcal{J} \models_{\mathfrak{B}} \psi$.

Applying Proposition 6.1.23 and Corollary 4.2.24 to assumption $\mathcal{I} \models_{\mathfrak{B}} \varphi$ yields:

$$\mathcal{I} \mathcal{E}_{\mathcal{B}}^n[y \mapsto d] \models_{\mathfrak{B}} \text{Cnew}(\varphi, y) \land \overline{\varphi}^y.$$ (6.3)

Using Theorem 5.3.11, we deduce from 6.2 that:

$$\langle \text{new } y : u \text{ with } \mathcal{U}[x := y] \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle.$$ This implies, that the following transition exists:

$$\langle \mathcal{U}[x := y] \mid \mathcal{I} \mathcal{E}_{\mathcal{B}}^n[y \mapsto d] \rangle \xrightarrow{\mathcal{I} \mathcal{E}_{\mathcal{B}}^n[y \mapsto d]} \langle \emptyset \mid \mathcal{K} \rangle,$$ (6.4)

for some $\mathcal{K}$ such that $\mathcal{J} = \mathcal{K}[y \mapsto d']$ for some suitable $d'$. Now applying Definition 6.1.30 to assumption 6.1, fact 6.3, and transition 6.4 yields $\mathcal{K} \models_{\mathfrak{B}} \psi$. As $\mathcal{J} = \mathcal{K}[y \mapsto d']$ implies that $\mathcal{K} = \mathcal{J}[y \mapsto y^\mathcal{K}]$, we derive that $\mathcal{J}[y \mapsto y^\mathcal{K}] \models_{\mathfrak{B}} \psi$, from which we infer $\mathcal{J} \models_{\mathfrak{B}} \psi$, using Theorem 4.1.17. This we had to prove.

- **Norm.** Here we have that $\Phi$ is the formula:

$$\{\varphi\} \mathcal{U} \{\psi\}.$$ Assume that:

$$\{\varphi\} \text{Norm}(\mathcal{U}) \{\psi\}.$$ (6.5)

Furthermore, assume that $\mathcal{I} \models_{\mathfrak{B}} \varphi$ and:

$$\langle \mathcal{U} \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle.$$ Then, by Theorem 5.3.19:

$$\langle \text{Norm}(\mathcal{U}) \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle.$$ Combining this last transition with assumption 6.5 and the assumption that $\mathcal{I} \models_{\mathfrak{B}} \varphi$, gives $\mathcal{J} \models_{\mathfrak{B}} \psi$, which we had to show.

- **If.** In this case $\Phi$ is the correctness formula:

$$\{\varphi\} \mathcal{R} \{\psi\}.$$ Assume that:

$$\{\varphi \land \mathcal{R}??\} \mathcal{R}! \{\psi\}.$$ (6.6)
6.1. The proof system

Furthermore, assume that $\mathcal{I} \models \varphi$, and that:

$$\mathcal{I} \xrightarrow{\{R\}} \mathcal{J}.$$ 

This last transition is only possible if:

$$\langle R! \mid \mathcal{I} \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \mathcal{J} \rangle,$$

and if $\mathcal{I} \models R?$. So, in fact, we have $\mathcal{I} \models \varphi \land R?$. Applying Definition 6.1.30 to the fact we just found, correctness formula 6.6 and transition 6.7 yields $\mathcal{J} \models \varphi$, and we are done.

- **Ealg.** Here, $\Phi$ must be the formula:

$$\{\varphi\} \mathcal{R} \{\varphi \land \neg R_1? \land \ldots \land \neg R_n?\}$$

Note that, if $\Phi$ has been derived, then $\mathcal{R}$ has to be consistent with respect to $\varphi$. This implies that $\varphi \land R? \models \mathcal{C}(R!)$ for any rule $R \in \mathcal{R}$.

Assume that:

$$\{\varphi\} \{\varphi\} \ldots \{\varphi\} \{\varphi\}.$$ 

Furthermore, assume that $\mathcal{I} \models \varphi$, and:

$$\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{K},$$

where $\mathcal{K}$ is a final state. This means there exists $m$ transitions:

$$\mathcal{I} \xrightarrow{\mathcal{R}} \mathcal{J}_1 \xrightarrow{\mathcal{R}} \ldots \xrightarrow{\mathcal{R}} \mathcal{J}_{m-1} \xrightarrow{\mathcal{R}} \mathcal{J}_m = \mathcal{K}.$$ 

Each transition is the result of applying one of the rules $R_i$ ($1 \leq i \leq n$). From our assumption that $\mathcal{I} \models \varphi$ and the correctness formula 6.8, it is very easy to deduce, by induction on $k$ ($1 \leq k \leq m$), that $\mathcal{J}_k \models \varphi$. This results in $\mathcal{K} \models \varphi$.

Moreover, since $\mathcal{K}$ is a final state, it is not possible that $\mathcal{K} \models R_i?$ for some $R_i$. Otherwise, we would have that $\mathcal{K} \models \varphi \land R_i?$ and also $\mathcal{K} \models \mathcal{C}(R_i!)$, by our observation at the beginning of this case. Then, by the Completeness Theorem 5.3.27, there would be a state $\mathcal{K}'$ such that:

$$\langle R_i! \mid \mathcal{K} \rangle \xrightarrow{\mathcal{K}'} \langle \emptyset \mid \mathcal{K}' \rangle,$$

contradicting our assumption that $\mathcal{K}$ is a final state. So, we deduce that $\mathcal{K} \models \neg R_1? \land \ldots \land \neg R_n?$. We already had found that $\mathcal{K} \models \varphi$, which means we are done.
- **Conseq.** Trivial.

- **Aux.** In this case $\Phi$ is the formula:

$$\{\emptyset\} \mathcal{R}^0 \{\xi\}.$$  

Assume that:

$$\{\emptyset \land \varphi\} \mathcal{R} \{\xi\},$$

such that the conditions of rule **Aux** are fulfilled. Furthermore, assume that $\mathcal{I}$ is a $\Sigma_e$-interpretation such that $\mathcal{I} \models_{\mathbf{B}} \theta$ and:

$$\mathcal{I} \xrightarrow{\mathcal{R}^0} \mathcal{K},$$

where $\mathcal{K}$ is a final state. We have to show that $\mathcal{K} \models_{\mathbf{B}} \xi$.

One of the conditions of **Aux** is that $\varphi$ has to be compatible with $\theta$. As $\theta \in \text{FORM}(\Sigma^\mathcal{S}_e)$, we have by Theorem 4.1.17 that $\mathcal{I} \models \Sigma^\mathcal{S}_e \models_{\mathbf{B}} \Sigma^\mathcal{S}_e \theta$.

According to Definition 6.1.25 this implies that there exist a $\Sigma^\mathcal{S}_e$-interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models_{\mathbf{B}} \theta \land \varphi$ and $\mathcal{I}' \models \Sigma^\mathcal{S}_e = \mathcal{I} \models \Sigma^\mathcal{S}_e$.

From transition 6.9 it follows that there exist a series of $m$ transitions:

$$\mathcal{I} = \mathcal{J}_0 \xrightarrow{\mathcal{R}^0} \mathcal{J}_1 \xrightarrow{\mathcal{R}^0} \ldots \xrightarrow{\mathcal{R}^0} \mathcal{J}_{m-1} \xrightarrow{\mathcal{R}^0} \mathcal{J}_m = \mathcal{K}.$$  

Since $\mathcal{R}$ contains auxiliary updates, it must be the case that for any $R \in \mathcal{R}$, there exist a $\mathcal{R}^0 \in \mathcal{R}^0$ such that $R$ contains auxiliary updates with respect to $\mathcal{R}^0$ (and vice versa). According to Definition 6.1.17 this means that $R^\mathcal{S}_e \equiv R^\mathcal{S}_e$, and that $\text{Cons}(R^\mathcal{S}_e, \mathcal{I})$ implies $\text{Cons}(R^\mathcal{S}_e, \mathcal{I})$, for any pair consisting of a $R \in \mathcal{R}$ and its corresponding $\mathcal{R}^0 \in \mathcal{R}^0$, and for any $\Sigma^\mathcal{S}_e$-interpretation $\mathcal{I}$.

We will show that these facts imply that there also exists a series of $m$ transitions starting from $\mathcal{I}'$:

$$\mathcal{I}' = \mathcal{J}'_0 \xrightarrow{\mathcal{R}} \mathcal{J}'_1 \xrightarrow{\mathcal{R}} \ldots \xrightarrow{\mathcal{R}} \mathcal{J}'_{m-1} \xrightarrow{\mathcal{R}} \mathcal{J}_m' = \mathcal{K},$$

such that $\mathcal{J}'_i \models \Sigma^\mathcal{S}_e$ and $\mathcal{J}_i \models \Sigma^\mathcal{S}_e$ are isomorphic for all $i$ ($0 \leq i \leq m$). The proof of this claim is by induction on $i$.

If $i = 0$, then there is nothing to prove. For the induction step, assume that $\mathcal{J}'_i \models \Sigma^\mathcal{S}_e$ and $\mathcal{J}_i \models \Sigma^\mathcal{S}_e$ are isomorphic ($i < m$). Let $R \in \mathcal{R}$, and let $R^\mathcal{S}_e \in \mathcal{R}^\mathcal{S}_e$ be the corresponding rule without auxiliary updates. We will first show that $R$ can be executed upon $\mathcal{J}'_i$, whenever $R^\mathcal{S}_e$ can be executed upon $\mathcal{J}_i$. So, suppose that $\mathcal{J}_i \models_{\mathbf{B}} R^\mathcal{S}_e$ and $\text{Cons}(R^\mathcal{S}_e, \mathcal{J}_i)$. We then have to show that $\mathcal{J}'_i \models_{\mathbf{B}} R^\mathcal{S}_e$ and $\text{Cons}(R^\mathcal{S}_e, \mathcal{J}'_i)$.
According to Definition 6.1.17 we have that \( R? \equiv R^0? \). Using the fact that \( R^0? \in \text{COND}(\Sigma_{\text{EA}}) \), the fact that \( \Sigma_{\text{EA}} \subseteq \Sigma_{\text{EA}}^R \subseteq \Sigma_{\text{EA}} \), and the fact that \( J'_i | \Sigma_{\text{EA}}^R \) and \( J_i | \Sigma_{\text{EA}} \) are isomorphic, we infer that \( J'_i \models_{\text{rR}} R? \). Moreover, since \( R^0? \in \text{UPD}(\Sigma_{\text{EA}}) \), we have by the same facts that \( \text{Cons}(R^0!, J'_i) \). From the fact that \( \Sigma_{\text{EA}} \subseteq \Sigma_{\text{EA}}^R \subseteq \Sigma_{\text{EA}} \), we infer that \( \text{Cons}(R^0!, J'_i | \Sigma_{\text{EA}}^R) \). Applying Definition 6.1.17 again, we conclude that \( \text{Cons}(R!, J'_i | \Sigma_{\text{EA}}) \), which implies \( \text{Cons}(R!, J'_i) \).

Now, assuming that \( J_{i+1} \) is the result of applying rule \( R \) to state \( J_i \), and that \( J'_{i+1} \) is the result of applying rule \( R^0 \) to \( J'_i \), we will show that \( J'_{i+1} | \Sigma_{\text{EA}}^R \) and \( J_{i+1} | \Sigma_{\text{EA}} \) are isomorphic. By our assumption the following transitions must exist:

\[
\langle R^0! | J_i \rangle \xrightarrow{J'_i} \langle \emptyset | J_{i+1} \rangle,
\]

and:

\[
\langle R! | J'_i \rangle \xrightarrow{J'_i} \langle \emptyset | J'_{i+1} \rangle.
\]

From transition 6.10 we deduce, using Theorem 6.1.28, that:

\[
\langle R^0! | J'_i | \Sigma_{\text{EA}}^R \rangle \xrightarrow{J'_i | \Sigma_{\text{EA}}^R} \langle \emptyset | J_{i+1} | \Sigma_{\text{EA}} \rangle.
\]

We will now prove by induction on the complexity \( \text{COMP}(R!) \) that:

\[
\langle R^0! | J'_i | \Sigma_{\text{EA}}^R \rangle \xrightarrow{J'_i | \Sigma_{\text{EA}}^R} \langle \emptyset | J'_{i+1} | \Sigma_{\text{EA}}^R \rangle.
\]

The idea is that taking the reduction \( | \Sigma_{\text{EA}}^R \) wipes out the effects of the auxiliary updates. Since by the induction hypothesis \( J'_i | \Sigma_{\text{EA}}^R \) and \( J_i | \Sigma_{\text{EA}} \) are isomorphic, we would have, by Theorem 5.3.23, that \( J'_{i+1} | \Sigma_{\text{EA}}^R \) and \( J_{i+1} | \Sigma_{\text{EA}} \) are isomorphic too.

For the base case, suppose that \( \text{COMP}(R!) = 0 \). In this case \( R! \) only contains simple updates. By the fact that \( \text{Cons}(R^0!, J'_i) \), it must be the case that \( \text{Cons}(R^0!, J'_i | \Sigma_{\text{EA}}^R) \). Applying Theorem 5.3.27, we infer that there exists a transition:

\[
\langle R^0! | J'_i | \Sigma_{\text{EA}}^R \rangle \xrightarrow{J'_i | \Sigma_{\text{EA}}^R} \langle \emptyset | J''_{i+1} \rangle.
\]

Applying Lemma 5.3.6 to this transition yields that \( J''_{i+1} \) can be written as:

\[
J''_{i+1} = (J'_i | \Sigma_{\text{EA}}^R) O_1 \ldots O_n,
\]

where each \( O_j \) corresponds to a simple update in \( R^0! \) which is interpreted with respect to evaluation state \( J'_i | \Sigma_{\text{EA}}^R \). By applying the same lemma to transition 6.11 we deduce that \( J'_{i+1} \) can be written as:

\[
J'_{i+1} = J'O_1 \ldots O_n M_1 \ldots M_k,
\]
where each $O'_j$ corresponds to the same update as $O_j$ does, and where each $M_i$ corresponds to an auxiliary update. Each $O'_j$ is interpreted with respect to evaluation state $J'_i$. Since $U \in \text{UPD}(\Sigma_{EA})$ for each $U \in R^n$, and since $\Sigma_{EA} \subseteq \Sigma_E$, we deduce that $O'_j = O_j$ for all $j$ ($1 \leq j \leq n$). On the other hand, by Definition 6.1.19, it is obvious that none of the operators $M_i$ is defined on $J'_i \mid \Sigma_E^-$. Combining these facts yields:

\[
J'_{i+1} \mid \Sigma_E^- = \left( J'_i \mid \Sigma_E^- \right) O_1 \ldots O_n M_1 \ldots M_k \mid \Sigma_E^- \\
= \left( J'_i \mid \Sigma_E^- \right) O_1 \ldots O_n \\
= J''_{i+1},
\]

which completes the proof for the base case $\text{COMP}(R!) = 0$.

For the induction step, assume that $\text{COMP}(R!) = n + 1$. For the sake of simplicity, we will assume that $R!$ contains only one extension update $U$ such that $\text{COMP}(\{U\}) = n + 1$ (the general case can easily be proven by induction). Let $U \equiv \text{new } x: u$ with $\mathcal{V}$. In this case $R!$ can be written as $R! = U \cup \{U\}$, where $U \not\in U$. Now, the following transitions must exist for $R!$:

\[
\langle U \cup \{\text{new } x: u \text{ with } \mathcal{V} \} \mid J'_i \rangle \xrightarrow{J'_i} \langle \text{new } x: u \text{ with } \mathcal{V} \mid K_1 \rangle \\
\xrightarrow{J'_i} \langle \emptyset \mid J'_{i+1} \rangle.
\]

Let $U^0$ be the result of deleting the auxiliary updates from $U$. Likewise, we define $V^0$. By the induction hypothesis we then have that:

\[
\langle U^0 \mid J'_i \mid \Sigma_E^- \rangle \xrightarrow{J'_i \mid \Sigma_E^-} \langle \emptyset \mid K_1 \mid \Sigma_E^- \rangle. \tag{6.13}
\]

The following transition must exist as well:

\[
\langle \mathcal{V} \mid K_1 E_d^n[x\rightarrow d] \rangle \xrightarrow{K_1 E_d^n[x\rightarrow d]} \langle \emptyset \mid K_2 \rangle,
\]

such that $J'_{i+1} = \mathcal{K}_2[x\rightarrow d']$ for some suitable $d'$. Applying the induction hypothesis to this transition yields:

\[
\langle \mathcal{V}^0 \mid (K_1 E_d^n[x\rightarrow d]) \mid \Sigma_E^- \rangle \\
\xrightarrow{(K_1 E_d^n[x\rightarrow d]) \mid \Sigma_E^-} \langle \emptyset \mid K_2 \mid \Sigma_E^- \rangle. \tag{6.14}
\]

At this point we consider two possibilities: either $u \in \text{DSORT}(\Sigma_{EA})$, or $u \in \text{DSORT}(\Sigma_{EA}^\text{ext}) \setminus \text{DSORT}(\Sigma_{EA})$.

In the first case, we deduce from transition 6.14 that:

\[
\langle \mathcal{V}^0 \mid (K_1 \mid \Sigma_E^-) E_d^n[x\rightarrow d] \rangle \xrightarrow{(K_1 \mid \Sigma_E^-) E_d^n[x\rightarrow d]} \langle \emptyset \mid K_2 \mid \Sigma_E^- \rangle.
\]
6.1. The proof system

From this transition we may infer:

\[
\langle \text{new } x : u \text{ with } \mathcal{V}^0 \mid (\mathcal{K}_1 \mid \Sigma_{\overline{e}}) \rangle \\
\xrightarrow{(\mathcal{K}_2 \mid \Sigma_{\overline{e}})[x \rightarrow d'']} \langle \emptyset \mid (\mathcal{K}_2 \mid \Sigma_{\overline{e}})[x \rightarrow d''] \rangle,
\]

for some suitable \( d'' \). It is easy to see that \( d'' = d' \), and:

\[
(\mathcal{K}_2 \mid \Sigma_{\overline{e}})[x \rightarrow d'] = (\mathcal{K}_2[x \rightarrow d']) \mid \Sigma_{\overline{e}} = \mathcal{J}_t' \mid \Sigma_{\overline{e}}.
\]

Moreover, we have \( \mathcal{R}^0! \equiv \mathcal{U}^0 \cup \{\text{new } x : u \text{ with } \mathcal{V}^0\} \). Combining transitions 6.13 and 6.15, and the fact we just found, yields:

\[
\langle \mathcal{R}^0! \mid \mathcal{J}_t' \mid \Sigma_{\overline{e}} \rangle \xrightarrow{\mathcal{J}_t' \mid \Sigma_{\overline{e}}} \langle \emptyset \mid \mathcal{J}_{t+1}' \mid \Sigma_{\overline{e}} \rangle.
\]

In the second case, where \( u \in \text{DSORT}(\Sigma_{\overline{e}}) \setminus \text{DSORT}(\Sigma_{\overline{e}}) \), we have that \( \mathcal{R}^0! = \mathcal{U}^0 \). Since, in this case, the extension update \( \mathcal{U} \equiv \text{new } x : u \text{ with } \mathcal{V} \) only contains auxiliary updates, it is routine to prove by induction on the complexity of \( \{\mathcal{U}\} \) that \( \mathcal{J}_{t+1}' \mid \Sigma_{\overline{e}} = \mathcal{K}_1 \mid \Sigma_{\overline{e}} \), from which the claim for this case follows.

What we now have proven is that:

\[
\langle \mathcal{R}^0! \mid \mathcal{J}_t' \mid \Sigma_{\overline{e}} \rangle \xrightarrow{\mathcal{J}_t' \mid \Sigma_{\overline{e}}} \langle \emptyset \mid \mathcal{J}_{t+1}' \mid \Sigma_{\overline{e}} \rangle,
\]

which implies, as we explained above, that \( \mathcal{J}_{t+1}' \mid \Sigma_{\overline{e}} \) and \( \mathcal{J}_{t+1} \mid \Sigma_{\overline{e}} \) are isomorphic. This completes the induction over \( m \), the length of the transition \( \mathcal{I} \xrightarrow{R} \mathcal{K} \). We conclude that \( \mathcal{K}' \mid \Sigma_{\overline{e}} \) and \( \mathcal{K} \mid \Sigma_{\overline{e}} \) are isomorphic. Furthermore, it is easy to see that \( \mathcal{K}' \) is final state, whenever \( \mathcal{K} \) is.

By our assumption that \( \{\theta \land \varphi\} \mathcal{R} \{\xi\} \), the fact that \( \mathcal{I} \models_{\mathfrak{B}} \theta \land \varphi \), and the claim we just proved, we deduce that \( \mathcal{K}' \models_{\mathfrak{B}} \xi \), where \( \mathcal{K}' \mid \Sigma_{\overline{e}} \) and \( \mathcal{K} \mid \Sigma_{\overline{e}} \) are isomorphic. Since \( \xi \in \text{FORM}_{\overline{e}} \), we also have \( \mathcal{K} \models_{\mathfrak{B}} \xi \).

This completes the proof of the Soundness Theorem 6.1.31. ■

6.1.32 COROLLARY Let \( \{\varphi\} \mathcal{R} \{\psi\} \) be a correctness formula such that:

\[
\models_{\mathfrak{P}} \{\varphi\} \mathcal{R} \{\psi\}.
\]

Then the evolving algebra \( \mathcal{R} \) is partially correct with respect to precondition \( \varphi \) and postcondition \( \psi \).

PROOF By Definition 6.1.30 and Theorem 6.1.31. ■
In the following theorem we will exhibit a counter example which shows that the system \( \mathcal{P} \) is incomplete. The incompleteness of \( \mathcal{P} \) is caused by the rule for contraction updates. In order to remove an element safely the condition \( \text{Crem} (\ldots) \) should be fulfilled, which expresses that there are no ‘pointers’ to the elements to be removed. However, the semantics of the remove operator \( \text{D} \) is defined purely semantically. There is only one check: whether the element to be removed actually exists.

**6.1.33 Theorem Incompleteness** \( \mathcal{P} \)

*The system \( \mathcal{P} \) is not complete, i.e. there is a correctness formula \( \Phi \) such that:

\[
\models_{\mathcal{P}} \Phi \quad \& \quad \not\vdash_{\mathcal{P}} \Phi.
\]

**Proof** Let \( \Sigma_{\mathcal{E}_{A}} \) be such that \( \text{DSORT} = \{ u \} \), and \( \text{SYM} = \{ a, b, c \} \) where \( a, b, c : u \). Define \( \Phi \) as follows:

\[
\Phi \equiv \{ a = b \land \downarrow c \land a \neq c \} \text{ rem } a : u \{ \downarrow b \land \downarrow c \}.
\]

We clearly have that \( \models_{\mathcal{P}} \Phi \). Suppose that \( \vdash_{\mathcal{P}} \Phi \) holds. Then, there should exist a formula \( \varphi \) such that the following two theorems and the following correctness formula are valid:

\[
a = b \land \downarrow c \land a \neq c \models \text{Crem}(\varphi, \{a\}) \land \varphi[a:=\bot]^{a}, \quad (6.16)
\]

\[
\{ \text{Crem}(\varphi, \{a\}) \land \varphi[a:=\bot]^{a} \} \text{ rem } a : u \{ \varphi \},
\]

\[
\varphi \models \not\downarrow b \land \downarrow c. \quad (6.17)
\]

(Note that \( \text{C}(\text{rem } a : u) = \downarrow a \) can be omitted since \( \text{Crem}(\varphi, \{a\}) \not\vdash \downarrow a \).)

We claim that the constant symbol \( b \) must occur in \( \varphi \). If this is indeed the case, we would have, by Definition 6.1.8, that \( \text{Crem}(\varphi, \{a\}) \not\vdash b \neq a \). This would contradict 6.16.

In order to obtain a contradiction, suppose that \( b \) does not occur in \( \varphi \). Let \( \mathcal{I} \) be a \( \Sigma \)-interpretation with the property that \( \mathcal{I} \models \varphi \). Such an interpretation exists, since otherwise we would have that \( \models \varphi \leftrightarrow \text{ff} \), in which case we would also have that \( \models \text{Crem}(\varphi, \{a\}) \land \varphi[a:=\bot]^{a} \leftrightarrow \text{ff} \), as can easily be verified using Theorem 6.1.12. This would imply that \( a = b \land \downarrow c \land a \neq c \) is inconsistent, which is clearly not the case.

By 6.17 and by the fact that \( \mathcal{I} \models \varphi \), we have that \( \mathcal{I} \models \not\downarrow b \land \downarrow c \). Since \( \mathcal{I} \models \downarrow c \), we deduce that \( |\mathcal{I}|_{\mathcal{U}} \neq \emptyset \). Define \( \mathcal{J} \) to be the \( \Sigma \)-interpretation, which is identical to \( \mathcal{I} \), except that \( b^{\mathcal{J}} = c^{\mathcal{J}} \), i.e. \( \mathcal{J} = \mathcal{I}[b:=c] \). By Theorem 4.1.17 and the assumption that \( b \) does not occur in \( \varphi \), we derive that \( \mathcal{J} \models \varphi \). But in that case, by 6.17 again, we also have \( \mathcal{J} \models \not\downarrow b \), which clearly is a contradiction. We conclude that the system \( \mathcal{P} \) is not complete. \( \blacksquare \)

**6.1.34 Remark** It might be the case that the system is complete in the sense of Cook (see e.g. [Apt81]) for EAs without contraction updates. This could be a topic for future research.
Up to now, we have considered only partial correctness. Suppose that we have
derived a correctness formula \( \{ \varphi \} \mathcal{R} \{ \psi \} \). What do we know in that case? We
only know then that if \( \mathcal{R} \) terminates, the end state must satisfy the postcondition \( \psi \), provided the begin state satisfied the precondition \( \varphi \).

In general we are also interested in the termination behaviour of an EA. A
proof system in which termination can be derived is called a proof system for
total correctness. It will turn out that a simple adaptation of the rule \textbf{Ealg} is
sufficient to prove the total correctness of EAs.

6.1.35 Definition Let \( N \in \text{Sort}(\Sigma^*_E) \), let \( \geq, \leq \in \text{Pred}_{\mathbb{N}}(\Sigma^*_E) \), and let the
interpretation of this sort and these symbols in \( \mathcal{B} \) be standard, i.e. as in
arithmetic. Moreover, let \( \varphi \in \text{FORM}(\Sigma_E) \), and let \( \mathcal{R} = \{ R_1, \ldots, R_n \} \subseteq \text{fin}
\text{Rule}(\Sigma^*_{EA}) \). The axiomatic semantics \( \mathcal{T} \) for total correctness of evolving
algebras is defined by the axiom \textbf{Locrem}, the rules \textbf{Ext}, \textbf{Norm}, \textbf{If}, \textbf{Conseq},
and \textbf{Aux} of Definition 6.1.26, and the following derivation rule:

**Termination rule for EAs**

\[
\begin{align*}
\varphi \models_{\mathcal{B}} t & \geq 0 \\
\{ \varphi \} R_1 \{ \varphi \} \quad \{ \varphi \wedge t = n \} R_1 \{ t < n \} \\
\vdots & \quad \vdots \\
\text{Term} : & \quad \{ \varphi \} R_n \{ \varphi \} \quad \{ \varphi \wedge t = n \} R_n \{ t < n \} \\
& \quad \{ \varphi \} \mathcal{R} \{ \varphi \wedge \neg R_1 \wedge \ldots \wedge \neg R_n \}
\end{align*}
\]

where \( t \in \text{TERM}_{\mathbb{N}}(\Sigma_E) \), where \( n \in \text{CON}_{\mathbb{N}}(\Sigma_E) \) is such that \( n \) does
not occur in \( t \), \( \varphi \) or \( \mathcal{R} \), and where \( \mathcal{R} \) is consistent relative to \( \varphi \).

6.1.36 Definition Let \( \Phi = \{ \varphi \} \Theta \{ \psi \} \) be a correctness formula. Then \( \Phi \) is
valid in the sense of total correctness, notation \( \models_{\mathcal{T}} \Phi \), if one of the following
condition holds:

- \( \Theta \subseteq \text{fin UPD}(\Sigma^*_{EA}) \), and for all standard \( \Sigma_E \)-interpretations \( \mathcal{I} \):

\[
\mathcal{I} \models_{\mathcal{B}} \varphi \Rightarrow \exists \mathcal{J} \cdot [ (\Theta \upharpoonright \mathcal{I}) \xrightarrow{\mathcal{I}} \{ \emptyset \mid \mathcal{J} \} \wedge \mathcal{J} \text{ is final} \wedge \mathcal{J} \models_{\mathcal{B}} \psi ].
\]

- \( \Theta \in \text{Rule}(\Sigma^*_{EA}) \), and for all standard \( \Sigma_E \)-interpretations \( \mathcal{I} \):

\[
\mathcal{I} \models_{\mathcal{B}} \varphi \Rightarrow \exists \mathcal{J} \cdot [ (\Theta \upharpoonright \mathcal{I}) \xrightarrow{\Theta} \mathcal{J} \wedge \mathcal{J} \text{ is final} \wedge \mathcal{J} \models_{\mathcal{B}} \psi ].
\]

- \( \Theta \subseteq \text{fin Rule}(\Sigma^*_{EA}) \), and for all standard \( \Sigma_E \)-interpretations \( \mathcal{I} \):

\[
\mathcal{I} \models_{\mathcal{B}} \varphi \Rightarrow \exists \mathcal{J} \cdot [ (\Theta \upharpoonright \mathcal{I}) \xrightarrow{\Theta} \mathcal{J} \wedge \mathcal{J} \text{ is final} \wedge \mathcal{J} \models_{\mathcal{B}} \psi ].
\]

In this case, \( \Theta \) is called totally correct with respect to precondition \( \varphi \)
and postcondition \( \psi \).
6.1.37 Definition If a derivation of a correctness formula \( \Phi \) in system \( T \) exists, then \( \Phi \) is called derivable or provable in \( T \), notation \( \vdash_T \Phi \).

6.1.38 Theorem Soundness \( T \)
For all correctness formulae \( \Phi \):
\[
\vdash_T \Phi \Rightarrow \models_T \Phi.
\]

Proof The proof follows the proof of Theorem 6.1.31 for the axiom LocRem, and the rules Ext, Norm, If, Conseq, and Aux. For this axiom and these rules the proof can easily be adapted to cover termination. The reason for this is that in the rule Locrem consistency is built in.

We only have to check the rule Term. Suppose that the theorem holds for the correctness formulae among the hypotheses of the rule. Then, by Theorem 6.1.31, we know that the postcondition \( \varphi \land \neg R_1 \land \ldots \land \neg R_n \) is true in a final state, provided that such a final state exists, and that \( \varphi \) is satisfied by the initial state of the run. This means that we have only to prove that \( R \) terminates. So, suppose that there exists a non-terminating run:
\[
I_0 \xrightarrow{R} I_1 \xrightarrow{R} \ldots \xrightarrow{R} I_n \xrightarrow{R} I_{n+1} \xrightarrow{R} \ldots,
\]
where \( I_0 \models_\mathfrak{S} \varphi \). By the first hypothesis of our rule Term, we then have that \( I_0 \models_\mathfrak{S} t \geq 0 \). This means that \( t \) has a definite value \( n \in \mathbb{N} \). Suppose that \( I_n \models_\mathfrak{S} \varphi \), and that rule \( R \in R \) is being executed. We then infer from hypothesis \( \{ \varphi \} R \{ \varphi \} \) that:
\[
I_n \xrightarrow{R} I_{n+1}.
\]
(6.18)
We will show that \( t^{I_{n+1}} < t^{I_n} \). Define \( J_n = I_n[n \mapsto t^{I_n}] \). From the assumption that \( n \) does not occur in \( t \) or \( \varphi \), we infer that \( t^{J_n} = t^{I_n} \) and:
\[
J_n \models_\mathfrak{S} \varphi \land t = n.
\]
From hypothesis \( \{ \varphi \land t = n \} \{ t < n \} \), we deduce that there exists a state \( J_{n+1} \) such that \( J_{n+1} \models_\mathfrak{S} t < n \), and such that:
\[
J_n \xrightarrow{R} J_{n+1}.
\]
(6.19)
Now, since \( n \) does not occur in \( t \), \( \varphi \) or \( R \), and since \( \mathbb{N} \) belongs to the static part of \( \Sigma_\mathcal{E} \), it can be easily shown that \( n^{J_{n+1}} = n^{J_n} \) (the proof of this fact is analogous to the proof of Lemma 5.3.7). Moreover, we infer that \( t^{J_{n+1}} < t^{J_n} \).

Let \( \Sigma'_\mathcal{E} \) be the signature which is equal to \( \Sigma_\mathcal{E} \) except that \( n \notin CON_\mathbb{N}(\Sigma'_\mathcal{E}) \). Then, applying Theorem 6.1.28 to transitions 6.18 and 6.19 yields:
\[
I_n \models \Sigma'_\mathcal{E} \xrightarrow{R} I_{n+1} \models \Sigma'_\mathcal{E},
\]
\[
J_n \models \Sigma'_\mathcal{E} \xrightarrow{R} J_{n+1} \models \Sigma'_\mathcal{E}.
\]
By the Soundness Theorem 5.3.26, and the fact that \( J_n \models \Sigma'_\mathcal{E} = I_n \models \Sigma'_\mathcal{E} \), we deduce from these transitions that \( J_{n+1} \models \Sigma'_\mathcal{E} \) and \( I_{n+1} \models \Sigma'_\mathcal{E} \) are isomorphic. This
implies that \( t^I_{n+1} = t^J_{n+1} \). We already saw that \( t^J_n = t^I_n \) and \( t^J_{n+1} < t^J_n \). We conclude that, going from intermediary state \( I_n \) to intermediary state \( I_{n+1} \), the value of \( t \) decreases: \( t^I_{n+1} < t^I_n \). The value \( t^I_n \) can, however, never become negative, since in each intermediary state \( I_n \) we have \( I_n \models \varphi \), and therefore \( I_n \models \exists \geq t \geq 0 \). So, the run must terminate.

\[ \quad \]

### 6.2 Example of a correctness proof

We will take up the example EA of Chapter 5 (see Example 5.1.18), and we will prove that this EA is partially and totally correct (with respect to a reasonable precondition and postcondition). Before doing this, we will define the signatures \( \Sigma_{EA} \), \( \Sigma_{EA}^\text{w} \), \( \Sigma_{E}^\text{u} \), \( \Sigma_{E}^\text{l} \), and the standard structure \( \mathcal{B} \) for \( \Sigma_{E}^\text{u} \). Then, we will present the EAs \( \mathcal{R}^0 \) and \( \mathcal{R} \), and formulate the correctness formulae we will derive. Furthermore, we will say something about the method by which we will construct the correctness proofs. Since this involves finding a suitable invariant, we will also shortly discuss the invariant we will use.

#### Signature \( \Sigma_{EA} \)

The following specifies the signature \( \Sigma_{EA} \) of the EA \( \mathcal{R}^0 \), i.e. the EA which we are primarily interested in.

\[
\begin{align*}
\text{SSORT}(\Sigma_{EA}) &= \{ \text{Char}, \text{Char}^* \}, \\
\text{DSORT}(\Sigma_{EA}) &= \{ \text{Stack} \}, \\
\text{SSYM}(\Sigma_{EA}) &= \{ \text{head}, \text{tail}, \text{nil}, \text{append} \}, \\
\text{DSYM}(\Sigma_{EA}) &= \{ \text{input}, \text{output}, \text{bottom}, \text{top}, \text{next}, \text{content} \}, \\
\text{head} & : \text{Char}^* \to \text{Char}, \\
\text{tail} & : \text{Char}^* \to \text{Char}^*, \\
\text{nil} & : \text{Char}^*, \\
\text{append} & : \text{Char}^* \times \text{Char} \to \text{Char}^*, \\
\text{input}, \text{output} & : \text{Char}^*, \\
\text{bottom}, \text{top} & : \text{Stack}, \\
\text{next} & : \text{Stack} \to \text{Stack}, \\
\text{content} & : \text{Stack} \to \text{Char}.
\end{align*}
\]

#### Signature \( \Sigma_{EA}^\text{w} \)

In order to prove the correctness of the EA \( \mathcal{R}^0 \), we will prove the correctness of another EA, named \( \mathcal{R} \), which contains an auxiliary statement. For that reason, the signature \( \Sigma_{EA} \) has to be extended.

\[
\begin{align*}
\text{SSORT}(\Sigma_{EA}^\text{w}) &= \text{SSORT}(\Sigma_{EA}) \cup \{ \text{N} \}, \\
\text{DSORT}(\Sigma_{EA}^\text{w}) &= \text{DSORT}(\Sigma_{EA}),
\end{align*}
\]
\[
\begin{align*}
\text{SSYM}(\Sigma_{\text{EA}}^{\text{aux}}) &= \text{SSYM}(\Sigma_{\text{EA}}) \cup \{1, +\}, \\
\text{DSYM}(\Sigma_{\text{EA}}^{\text{aux}}) &= \text{DSYM}(\Sigma_{\text{EA}}) \cup \{\text{index}\}, \\
1 & : \mathbb{N}, \\
+ & : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, \\
\text{index} & : \text{Stack} \rightarrow \mathbb{N}.
\end{align*}
\]

**Signature** \(\Sigma_\delta\)

The signature \(\Sigma_\delta\) contains all symbols which may occur in preconditions and postconditions of correctness formulae.

\[
\begin{align*}
\text{SORT}(\Sigma_\delta) &= \text{SSORT}(\Sigma_{\text{EA}}^{\text{aux}}) \cup \text{DSORT}(\Sigma_{\text{EA}}^{\text{aux}}), \\
\text{SORT}(\Sigma_\delta^s) &= \text{SSORT}(\Sigma_{\text{EA}}^{\text{aux}}), \\
\text{SYM}(\Sigma_\delta) &= \text{SSYM}(\Sigma_{\text{EA}}^{\text{aux}}) \cup \text{DSYM}(\Sigma_{\text{EA}}^{\text{aux}}) \\
&\quad \cup \{at, from, length, 0, -, \geq, <, n\}, \\
\text{SYM}(\Sigma_\delta^s) &= \text{SSYM}(\Sigma_{\text{EA}}^{\text{aux}}) \cup \{at, from, length, 0, -, \geq, <\}, \\
at & : \text{Char}^* \times \mathbb{N} \rightarrow \text{Char}, \\
from & : \text{Char}^* \times \mathbb{N} \rightarrow \text{Char}^*, \\
\text{length} & : \text{Char}^* \rightarrow \mathbb{N}, \\
0 & : \mathbb{N}, \\
- & : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, \\
\geq, < & : \mathbb{N} \times \mathbb{N}, \\
\bar{n} & : \mathbb{N}.
\end{align*}
\]

**Standard structure** \(\mathcal{B}\) for \(\Sigma_\delta^s\)

The following speaks for itself:

\[\begin{align*}
\text{N} & \text{ denotes the set of natural numbers,} \\
\text{Char} & \text{ denotes the set of characters (e.g. the ASCII-set),} \\
\text{Char}^* & \text{ denotes the set of all strings over Char with finite length,} \\
0, 1, +, - & \text{ denote the usual arithmetical constants and operations, where} \\
& \text{ - is treated as a partial function (e.g. } 0 - 1 \text{ is undefined),} \\
\geq, < & \text{ denote the usual arithmetical relations,} \\
\text{nil, head, tail} & \text{ denote the usual constant and functions on strings, where head} \\
& \text{ and tail are treated as partial operations,} \\
\text{append}(s, c) & \text{ denotes the string which is the result of appending the} \\
& \text{character c at the right end of string s,}
\end{align*}\]
6.2. Example of a correctness proof

length(s) denotes the length of string s,

at(s, n) denotes the nth character in string s, if 1 ≤ n ≤ length(s), otherwise undefined,

from(s, n) denotes the right substring of s starting at the nth character of s, if 1 ≤ n ≤ length(s), otherwise undefined.

Signature \( \Sigma_\varepsilon \)

When the correctness of the EA \( \mathcal{R} \) has been demonstrated, we want to establish the correctness of \( \mathcal{R}^0 \), which does not contain auxiliary updates. The signature \( \Sigma_\varepsilon \) contains all symbols that may occur in the precondition and postcondition of the correctness formula for \( \mathcal{R}^0 \).

\[
\begin{align*}
\text{SORT}(\Sigma_\varepsilon) &= \text{SORT}(\Sigma_\varepsilon), \\
\text{SYM}(\Sigma_\varepsilon) &= \text{SYM}(\Sigma_\varepsilon) \setminus \{\text{index}\}.
\end{align*}
\]

Evolving algebra \( \mathcal{R}^0 \)

For convenience we reprint the EA \( \mathcal{R}^0 \) which we wish to prove to be correct. It reads the input string input, pushes the characters on a stack Stack, and then reads the stack and produces the output string output which is the reverse of the input. The stack is implemented by the dynamic universe Stack in which new 'nodes' are added and connected by the function next. The bottom of the stack is marked by bottom, whereas the current top is called top. The function content specifies the character which is stored at a node of the stack. Remember that all updates are executed in parallel!

\( R_1^0 \equiv \) if input ≠ nil then

new \( x : \text{Stack} \) with

\[
\begin{align*}
top &:= x \\
\text{next}(x) &:= \text{top} \\
\text{content}(x) &:= \text{head}(\text{input}) \\
\text{input} &:= \text{tail}(\text{input})
\end{align*}
\]

\( R_2^0 \equiv \) if input = nil ∧ top ≠ bottom then

\[
\begin{align*}
\text{output} &:= \text{append}(\text{output}, \text{content}(\text{top})) \\
\text{top} &:= \text{next}(\text{top}) \\
\text{rem} &\top : \text{Stack}
\end{align*}
\]

Evolving algebra \( \mathcal{R} \)

If one tries to prove the original EA \( \mathcal{R}^0 \) to be correct, one encounters the problem that it is impossible (so it seems to the author) to relate the characters on the stack with the characters in the original input string. For that reason an auxiliary update is added to \( \mathcal{R}^0 \) which marks each node on the stack with an index number. For this purpose the function index is used.
\( R_1 \equiv \) if \( \text{input} \neq \text{nil} \) then

new \( x : \text{Stack} \) with
\begin{align*}
\text{top} & := x \\
\text{next}(x) & := \text{top} \\
\text{content}(x) & := \text{head}(\text{input}) \\
\text{index}(x) & := \text{index}(\text{top}) + 1 \\
\text{input} & := \text{tail}(\text{input})
\end{align*}

\( R_2 \equiv \) if \( \text{input} = \text{nil} \land \text{top} \neq \text{bottom} \) then

\( \text{output} := \text{append}(\text{output}, \text{content}(\text{top})) \)
\( \text{top} := \text{next}(\text{top}) \)
\( \text{rem top} : \text{Stack} \)

**Correctness formula for** \( \mathcal{R}^0 \)

We first define some formulae and provide them with a name, which we will use later.

\( \text{pre} \equiv \) \( \text{input} = \text{string} \land \text{output} = \text{nil} \land \text{top} = \text{bottom} \).

\( \text{post} \equiv \) \( \text{length}(\text{output}) = \text{length}(\text{string}) \land \forall m : \mathbb{N} \cdot [m \geq 1 \land m \leq \text{length}(\text{string}) \rightarrow \text{at}(\text{output}, m) = \text{at}(\text{string}, \text{length}(\text{string}) + 1 - m)] \).

The correctness formula we are going to prove is:

\( \{ \text{pre} \} \mathcal{R}^0 \{ \text{post} \} \).

**Correctness formula for** \( \mathcal{R} \)

Define the following formula, which formalizes the initialization of the auxiliary function \( \text{index} \).

\( \text{aux} \equiv \) \( \text{index}(\text{bottom}) = 0 \land \forall y : \text{Stack} \cdot [y \neq \text{bottom} \rightarrow \neg \downarrow \text{index}(y)] \).

The correctness formula for \( \mathcal{R} \) is as follows:

\( \{ \text{pre} \land \text{aux} \} \mathcal{R} \{ \text{post} \} \).

**Method of proof**

There are two important methodological steps in finding a correctness proof of an EA. The first step involves finding a formula which represents the invariant of the EA. The formula \( \text{inv} \) is an invariant of the EA \( \mathcal{R} \), if for all rules \( R \in \mathcal{R} \) the following correctness formula is provable:

\( \{ \text{inv} \} R \{ \text{inv} \} \).
6.2. Example of a correctness proof

Moreover, the following theorems about the invariant must hold:

\[ \text{pre} \land \text{aux} \models_{\text{inv}} \]

and:

\[ \text{inv} \land \bigwedge_{R \in R} \neg \text{R} \models_{\text{post}} \]

The second step in finding the correctness proof consists of finding the basic steps in the proof, i.e. steps where the rule \textbf{LocRem} is used. We will locate these basic steps by \textit{backward reasoning}. This is done by inspecting the conclusion of a rule in order to infer which premiss has to be derived.

\textbf{The invariant}

The most complex part of finding a correctness proof is designing the \textit{invariant}. In this case, the invariant formalizes the stack structure. When we firstly read the \textit{input} and store it in the \textit{Stack}, we have to be sure that we can retrieve it correctly. In particular, we want to be able to retrieve all characters in the \textit{right order}. The invariant also contains information about the current \textit{top} of the stack. At any point of the run, if \textit{top} \neq \textit{bottom}, there may not be ‘pointers’ to \textit{top}, since the condition \textbf{Cre}m \textit{(inv, \{top\})} has to be true. Otherwise, we are not sure that removing the \textit{top} results in a sound data structure. This explains the complexity of the following invariant. (Note that we take the freedom to use abbreviations for quantifier expressions.)

\[ \text{inv} \equiv \]

\[ \downarrow \text{top} \land \downarrow \text{index(top)} \land \text{index(bottom)} = 0 \]

\[ \land \forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index(y)} \Rightarrow \downarrow \text{next(y)} \land \downarrow \text{index(next(y))} ] \]

\[ \land \forall y: \text{Stack} \cdot [\text{top} \neq \text{bottom} \Rightarrow \text{next(y)} \neq \text{top} ] \]

\[ \land \forall y, z, w: \text{Stack} \cdot [\text{next(y)} = z \land z \neq \text{bottom} \land \downarrow \text{index(z)} \land w \neq y \Rightarrow \text{next(w)} \neq z ] \]

\[ \land \text{index(top)} = \text{length(string)} - \text{length(input)} - \text{length(output)} \]

\[ \land \forall y, z: \text{Stack} \cdot [\text{next(y)} = z \land y \neq \text{bottom} \land \downarrow \text{index(y)} \Rightarrow \text{index(y)} = \text{index(z)} + 1 ] \]

\[ \land [\text{input} \neq \text{nil} \Rightarrow \text{input} = \text{from(string, index(top)+1)} ] \]

\[ \land [\text{input} \neq \text{nil} \Rightarrow \text{output} = \text{nil} ] \]

\[ \land \forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index(y)} \Rightarrow \text{content(y)} = \text{at(string, index(y))} ] \]

\[ \land \forall 1 \leq i \leq \text{length(string)} - \text{index(top)} \cdot [\text{input} = \text{nil} \Rightarrow \text{at(output, i)} = \text{at(string, length(string) - i+1)} ] . \]
Partial correctness proof of $\mathcal{R}^0$

In order to show that $inv$ really is an invariant, we have to prove the two correctness formulae \{\textit{inv}\} $R_1$ \{\textit{inv}\}, and \{\textit{inv}\} $R_2$ \{\textit{inv}\}. We firstly sketch the proof of the first formula. Let $U$ be the following set of updates:

\[
U \equiv \begin{align*}
top & := x \\
ext(x) & := top \\
content(x) & := head(input) \\
index(x) & := index(top) + 1 \\
input & := tail(input)
\end{align*}
\]

Clearly, new $x : \text{Stack}$ with $U$ is a normal form of $R_1!$. We will prove that $inv$ is invariant under $R_1$ by using $U$. Note that $C(U) \equiv \downarrow x$, and that $U$ can considered to be a maximal substitution. We will write it as $\sigma_U$.

If we inspect the rule If of system $\mathcal{P}$, we see that in order to prove the correctness formula \{\textit{inv}\} $R_1$ \{\textit{inv}\}, we firstly have to derive:

\[
\{R_1? \land inv\} R_1! \{inv\}.
\]

According to the rule Norm, it is sufficient to derive:

\[
\{R_1? \land inv\} \text{ new } x : \text{Stack with } U \{\text{inv}\}.
\]

But then, according rule Ext, we have to prove:

\[
\{C_{\text{new}}(R_1? \land \text{inv}, x) \land (R_1? \land \text{inv})^x\} U \{\text{inv}\}.
\] (6.20)

Since $U$ is a set of local function updates, we have to use the rule LocRem. There are no contraction updates in $U$, so the condition $C_{\text{rem}}(\cdot, \cdot)$ is vacuously true and the set $T_U$ is empty. Therefore, we have the following correctness formula for $U$:

\[
\{C(U) \land \text{inv } \sigma_U\} U \{\text{inv}\}.
\]

In order to apply Conseq and to conclude that correctness formula 6.20 holds, we have to prove the theorem:

\[
C_{\text{new}}(R_1? \land \text{inv}, x) \land (R_1? \land \text{inv})^x \models_{\mathfrak{B}} C(U) \land \text{inv } \sigma_U.
\]

In fact, we will not formally prove this theorem, but only list the formulae involved. The truth of the theorem is not very hard to see, albeit it time consuming for some details. Note that if $\varphi$ has the format $\forall y \cdot [\psi \rightarrow \chi]$, we immediately simplify $\varphi^x$ to $\forall y \cdot [y \neq x \land \psi \rightarrow \chi]$.

\[
R_1? \equiv input \neq \text{nil}.
\]
6.2. Example of a correctness proof

\( C_{\text{new}}(R_1 ? \land \text{inv}, x) \equiv \)
\[
\downarrow x \land \forall y: \text{Stack} \cdot \text{next}(y) \neq x \land \downarrow \text{content}(x) \land \downarrow \text{next}(x) \land \downarrow \text{index}(x) \\
\land \text{top} \neq x \land \text{bottom} \neq x.
\]
\( (R_1 ? \land \text{inv})^x \equiv \)
\[
\text{input} \neq \text{nil} \\
\land \downarrow \text{top} \land \downarrow \text{index}(\text{top}) \land \text{index}(\text{bottom}) = 0 \\
\land \forall y: \text{Stack} \cdot [y \neq x \land y \neq \text{bottom} \land \downarrow \text{index}(y) \\
\rightarrow \downarrow \text{next}(y) \land \downarrow \text{index}(\text{next}(y))]
\]
\[
\land \forall y: \text{Stack} \cdot [y \neq x \land \text{top} \neq \text{bottom} \rightarrow \text{next}(y) \neq \text{top}] \\
\land \forall y, z, w: \text{Stack} \cdot [y \neq x \land z \neq x \land w \neq x \land \text{next}(y) = z \land z \neq \text{bottom} \\
\land \downarrow \text{index}(z) \land w \neq y \rightarrow \text{next}(w) \neq z]
\]
\[
\land \text{index}(\text{top}) = \text{length}(\text{string}) - \text{length}(\text{input}) - \text{length}(\text{output}) \\
\land \forall y, z: \text{Stack} \cdot [y \neq x \land z \neq x \land \text{next}(y) = z \land y \neq \text{bottom} \land \downarrow \text{index}(y) \\
\rightarrow \text{index}(y) = \text{index}(z) + 1]
\]
\[
\land [\text{input} \neq \text{nil} \rightarrow \text{input} = \text{from}(\text{string}, \text{index}(\text{top}) + 1)] \\
\land [\text{input} \neq \text{nil} \rightarrow \text{output} = \text{nil}]
\]
\[
\land \forall y: \text{Stack} \cdot [y \neq x \land y \neq \text{bottom} \land \downarrow \text{index}(y) \\
\rightarrow \text{content}(y) = \text{at}(\text{string}, \text{index}(y))]
\]
\[
\land \forall 1 \leq i \leq \text{length}(\text{string}) - \text{index}(\text{top}) \cdot [\text{input} = \text{nil} \\
\rightarrow \text{at}(\text{output}, i) = \text{at}(\text{string}, \text{length}(\text{string}) - i + 1)].
\]

\( C(\mathcal{U}) \equiv \downarrow x. \)

\( \text{inv } \sigma_\mathcal{U} \equiv \)
\[
\downarrow x \land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](x) \\
\land \text{index}[x \mapsto \text{index}(\text{top}) + 1](\text{bottom}) = 0 \\
\land \forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](y) \\
\rightarrow \downarrow \text{next}[x \mapsto \text{top}](y) \land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](\text{next}[x \mapsto \text{top}](y))]
\]
\[
\land \forall y: \text{Stack} \cdot [x \neq \text{bottom} \rightarrow \text{next}[x \mapsto \text{top}](y) \neq x] \\
\land \forall y, z, w: \text{Stack} \cdot [\text{next}[x \mapsto \text{top}](y) = z \land z \neq \text{bottom} \\
\land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](z) \land w \neq y \rightarrow \text{next}[x \mapsto \text{top}](w) \neq z]
\]
\[
\land \text{index}[x \mapsto \text{index}(\text{top}) + 1](x) \\
= \text{length}(\text{string}) - \text{length}(\text{tail}(\text{input})) - \text{length}(\text{output}) \\
\land \forall y, z: \text{Stack} \cdot [\text{next}[x \mapsto \text{top}](y) = z \land y \neq \text{bottom} \\
\land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](y) \\
\rightarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](y) = \text{index}[x \mapsto \text{index}(\text{top}) + 1](z) + 1]
\[ \land [\text{tail}(\text{input}) \neq \text{nil} \rightarrow \text{tail}(\text{input}) = \text{from}(\text{string}, \text{index}[x \mapsto \text{index}(\text{top}) + 1](x) + 1)] \]

\[ \land [\text{tail}(\text{input}) \neq \text{nil} \rightarrow \text{output} = \text{nil}] \]

\[ \land \forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](y) \rightarrow \text{content}[x \mapsto \text{head}(\text{input})](y) = \text{at}(\text{string}, \text{index}[x \mapsto \text{index}(\text{top}) + 1](y))] \]

\[ \land \forall 1 \leq i \leq \text{length}(\text{string}) \rightarrow \text{index}[x \mapsto \text{index}(\text{top}) + 1](x) \cdot [\text{tail}(\text{input}) = \text{nil} \rightarrow \text{at}(\text{output}, i) = \text{at}(\text{string}, \text{length}(\text{string}) - i + 1)]. \]

We will now concern ourselves with rule \( R_2 \). Like in the case for \( R_1 \), we have to prove the following correctness formula:

\[ \{ R_2? \land \text{inv} \} R_2! \{ \text{inv} \}. \]

In this case we have that \( \mathbf{C}(R_2!) \equiv \downarrow \text{top} \land \text{top} \neq \text{next}(\text{top}) \). Again, we want to apply rule \textbf{LocRem}. Since one contraction update is involved, the condition \textbf{Crem} \( \langle \text{inv}, \{ \text{top} \} \rangle \) is involved. Furthermore, since \( \text{top} := \text{next}(\text{top}) \) is an update in \( R_2! \), we do not have a substitution \( \sigma_{\mathcal{U},\bot} \). This leads to the following correctness formula:

\[ \{ \mathbf{C}(R_2!) \land \textbf{Crem} \langle \text{inv}, \{ \text{top} \} \rangle \land \overline{\text{inv}\sigma_{R_2!}}^{\text{top}} \} R_2! \{ \text{inv} \}. \]

But then we still have to prove the next theorem:

\[ R_2? \land \text{inv} \models_{\mathfrak{S}} \mathbf{C}(R_2!) \land \textbf{Crem} \langle \text{inv}, \{ \text{top} \} \rangle \land \overline{\text{inv}\sigma_{R_2!}}^{\text{top}}. \]

We will not prove this theorem, but only list the formulae involved.

\[ R_2? \equiv \quad \text{input} = \text{nil} \land \text{top} \neq \text{bottom}. \]

\[ \text{inv} \equiv \]

\[ \downarrow \text{top} \land \downarrow \text{index}(\text{top}) \land \text{index}(\text{bottom}) = 0 \]

\[ \land \forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index}(y) \rightarrow \downarrow \text{next}(y) \land \downarrow \text{index}(\text{next}(y)))] \]

\[ \land \forall y: \text{Stack} \cdot [\text{top} \neq \text{bottom} \rightarrow \text{next}(y) \neq \text{top}] \]

\[ \land \forall y, z, w: \text{Stack} \cdot [\text{next}(y) = z \land z \neq \text{bottom} \land \downarrow \text{index}(z) \land w \neq y \rightarrow \text{next}(w) \neq z] \]

\[ \land \text{index}(\text{top}) = \text{length}(\text{string}) - \text{length}(\text{input}) - \text{length}(\text{output}) \]

\[ \land \forall y, z: \text{Stack} \cdot [\text{next}(y) = z \land y \neq \text{bottom} \land \downarrow \text{index}(y) \rightarrow \text{index}(y) = \text{index}(z) + 1] \]

\[ \land [\text{input} \neq \text{nil} \rightarrow \text{input} = \text{from}(\text{string}, \text{index}(\text{top}) + 1)] \]

\[ \land [\text{input} \neq \text{nil} \rightarrow \text{output} = \text{nil}] \]
\[ \begin{align*}
\forall y: \text{Stack} \cdot [y \neq \text{bottom} \land \downarrow \text{index}(y) \\
\rightarrow \text{content}(y) = \text{at}(\text{string}, \text{index}(y))] \\
\forall 1 \leq i \leq \text{length(string)} - \text{index}(\text{top}) \cdot [\text{input} = \text{nil} \\
\rightarrow \text{at}(\text{output}, i) = \text{at}(\text{string}, \text{length(string)} - i + 1)].
\end{align*} \]

\[ \text{C(R2)} \equiv \downarrow \text{top} \land \text{top} \neq \text{next}(\text{top}). \]

\[ \text{Crem}((\text{inv}, \{\text{top}\}) =  \]
\[ \downarrow \text{top} \land \forall y: \text{Stack} \cdot [y \neq \text{top} \rightarrow \text{next}(y) \neq \text{top}] \land \text{bottom} \neq \text{top}. \]

\[ \overline{\text{inv} \sigma_{R2}} \equiv \]
\[ \downarrow \text{next}(\text{top}) \land \downarrow \text{index}(\text{next}(\text{top})) \land \text{index}(\text{bottom}) = 0 \]
\[ \forall y: \text{Stack} \cdot [y \neq \text{top} \land y \neq \text{bottom} \land \downarrow \text{index}(y) \\
\rightarrow \downarrow \text{next}(y) \land \downarrow \text{index}(\text{next}(y))]. \]
\[ \forall y: \text{Stack} \cdot [y \neq \text{top} \land \text{next}(\text{top}) \neq \text{bottom} \rightarrow \text{next}(y) \neq \text{next}(\text{top})] \]
\[ \forall y, z, w: \text{Stack} \cdot [y \neq \text{top} \land z \neq \text{top} \land w \neq \text{top} \land \text{next}(y) = z \\
\land z \neq \text{bottom} \land \downarrow \text{index}(z) \land w \neq y \rightarrow \text{next}(w) \neq z] \]
\[ \forall y: \text{Stack} \cdot [y \neq \text{top} \land \text{next}(\text{top}) = \text{length(string)} - \text{length(append(output, content(top)))} \\
\land \text{length(append(output, content(top))) = index(bottom)}] \]
\[ \forall y, z: \text{Stack} \cdot [y \neq \text{top} \land z \neq \text{top} \land \text{next}(y) = z \land y \neq \text{bottom} \\
\land \downarrow \text{index}(y) \rightarrow \text{index}(y) = \text{index}(z) + 1]. \]
\[ \forall y: \text{Stack} \cdot [y \neq \text{top} \land \text{next}(\text{top}) = \text{length(string)} + 1] \]
\[ \forall y: \text{Stack} \cdot [y \neq \text{top} \land \text{next}(\text{top}) = \text{content}(y) = \text{at}(\text{string}, \text{index}(y))] \\
\land \forall 1 \leq i \leq \text{length(string)} - \text{index}(\text{next}(\text{top})) \cdot [\text{input} = \text{nil} \\
\rightarrow \text{at}(\text{append(output, content(top)), i}) \\
= \text{at}(\text{string}, \text{length(string)} - i + 1)]. \]

Taking for granted the two ‘big’ theorems above, we see that inv indeed is invariant under \( R_1 \) and \( R_2 \). Applying rule \textbf{Ealg} then gives us:

\[ \{\text{inv}\} \; \text{R} \; \{\text{inv} \land \neg R_1? \land \neg R_2?\}. \]

Now, using the theorems \( \pre \land \text{aux} \models \text{inv} \) and \( \text{inv} \land \neg R_1? \land \neg R_2? \models \text{post} \), which are easy to verify, we infer that \( \{\pre \land \text{aux}\} \; \text{R} \; \{\text{post}\} \). Applying \textbf{Aux} to this formula finally gives us \( \{\pre\} \; \text{R}^0 \; \{\text{post}\} \). Resuming, the proof of partial correctness of \( \text{R}^0 \) is as follows:

1. \[ \text{Cnew}(R_1? \land \text{inv}, \text{x}) \land \overline{(R_1? \land \text{inv})^x} \models \quad \text{C(U)} \land \text{inv} \sigma_{U} \]
2. \[ \{\text{C(U)} \land \text{inv} \sigma_{U}\} \; \text{U} \; \{\text{inv}\} \quad \text{(LocRem)} \]
3. \( \{ \text{Cnew} (R_1 \land inv, x) \land (R_1 \land inv)^{\forall} \} \cup \{ \text{inv} \} \)  
   (Conseq, 1, 2)

4. \( \{ R_1 \land inv \} \text{ new } x : \text{Stack} \text{ with } \mathcal{U} \{ \text{inv} \} \)  
   (Ext, 3)

5. \( \{ R_1 \land inv \} R_1! \{ \text{inv} \} \)  
   (Norm, 4)

6. \( \{ \text{inv} \} R_1 \{ \text{inv} \} \)  
   (If, 5)

7. \( R_2 \land inv \models_{\exists} C(R_2!) \land \text{Crem} (\text{inv}, \{ \text{top} \}) \land inv_{R_2}^{\forall} \)  
   (LocRem)

8. \( \{ C(R_2!) \land \text{Crem} (\text{inv}, \{ \text{top} \}) \land inv_{R_2}^{\forall} \} R_2! \{ \text{inv} \} \)  
   (Conseq, 7, 8)

9. \( \{ R_2 \land inv \} R_2! \{ \text{inv} \} \)  
   (If, 9)

10. \( \{ \text{inv} \} R_2 \{ \text{inv} \} \)  
    (Ealg, 6, 10)

11. \( \{ \text{inv} \} \mathcal{R} \{ \text{inv} \land \neg R_1 \land \neg R_2 \} \)  
    (Ealg, 6, 10)

12. \( \text{pre} \land \text{aux} \models_{\exists} \text{inv} \)  
    (Term, 6, 10–13)

13. \( \text{inv} \land \neg R_1 \land \neg R_2 \models_{\exists} \text{post} \)  
    (Conseq, 14–16)

14. \( \{ \text{pre} \land \text{aux} \} \mathcal{R} \{ \text{post} \} \)  
    (Aux, 14)

15. \( \{ \text{pre} \} \mathcal{R}^0 \{ \text{post} \} \)  
    (Aux, 14)

**Total correctness proof of \( \mathcal{R}^0 \)**

In order to establish total correctness of \( \mathcal{R}^0 \) we need to prove the following additional premisses: the theorem \( \text{inv} \models_{\exists} t \geq 0 \), and the two correctness formulae \( \{ \text{inv} \land t = n \} R_1 \{ t < n \} \), and \( \{ \text{inv} \land t = n \} R_2 \{ t < n \} \). These are easily derived using the following expression for \( t \):

\[
 t \equiv \text{length}(\text{string}) + \text{length}(\text{input}) - \text{length}(\text{output}).
\]

The total correctness proof now follows steps 1 until 10 of the proof of the partial correctness, and from then it proceeds as follows:

11. \( \text{inv} \models_{\exists} t \geq 0 \)

12. \( \{ \text{inv} \land t = n \} R_1 \{ t < n \} \)  
   (\ldots)

13. \( \{ \text{inv} \land t = n \} R_2 \{ t < n \} \)  
   (\ldots)

14. \( \{ \text{inv} \} \mathcal{R} \{ \text{inv} \land \neg R_1 \land \neg R_2 \} \)  
   (Term, 6, 10–13)

15. \( \text{pre} \land \text{aux} \models_{\exists} \text{inv} \)  
   (Term, 6, 10–13)

16. \( \text{inv} \land \neg R_1 \land \neg R_2 \models_{\exists} \text{post} \)  
   (Term, 6, 10–13)

17. \( \{ \text{pre} \land \text{aux} \} \mathcal{R} \{ \text{post} \} \)  
   (Term, 6, 10–13)

18. \( \{ \text{pre} \} \mathcal{R}^0 \{ \text{post} \} \)  
   (Term, 6, 10–13)
Chapter 7

Conclusions and future work

We will summarize the results of this thesis with respect to our initial research program. We wanted to present a formalization of many-sorted evolving algebras. This formalization had to include a syntax, an operational semantics, and an axiomatic (Hoare-style) semantics.

Since an axiomatic semantics presupposes a logic to express preconditions and postconditions of evolving algebras, we firstly introduced a logic which also had to be many-sorted. In this $E$-logic we included partial functions, extended function symbols, and extended terms. Having read Chapter 6, the reader will understand why our $E$-logic should have this features: updates can cause functions to become undefined at certain arguments, whereas the update rule of the axiomatic semantics introduces extended terms in the precondition. Note that our treatment of the update rule by means of extended functions and terms differs from the usual one (see e.g. [AO91b]). Our notation is less general, but easier to read.

After having defined a Fitch-style natural deduction system for $E$-logic, we presented a semantics for it based on partial structures with possibly empty domains. This was a natural step, as static algebras can be seen as partial structures without predicates. As a result, structures of $E$-logic naturally contain states of evolving algebras as reducts. For this logic we have shown soundness and completeness, not by reducing it to another logic, but by directly carrying out the Lindenbaum-Henkin construction. As a preliminary to the results of Chapters 5 and 6, we introduced a number of operators which act on structures for $E$-logic, and proved a number of theorems about them. These operators were used in the rules of the operational semantics of evolving algebras. On the other hand, the theorems about these operators guided us to the construction of the axiomatic rule for sets of simple updates.

Having done the preliminary work on $E$-logic, we could start the actual work on evolving algebras. Using the syntax of this logic it was easy to define the syntax of evolving algebras. Far more interesting, of course, was the exercise of defining its operational semantics. Before this, we discussed the notion of consistency in connection to evolving algebras. This turned out to be easier
than expected. The result was a simple definition, which we derived from the intuitive notion of naive consistency.

The work on the structural operational semantics turned out to be more complicated than expected. Firstly, the rule for extension updates showed us to be careful with reserve elements. This is caused by the presence of the contraction update. We found that elements which are discarded by this rule, may not become reserve elements. Moreover, due to the contraction update again, it turned out not to be always possible to reset the extension variable to its original value, since it could be the case that this value had been removed by a contraction.

Secondly, in the proof of the Conversion Theorem (5.3.11), we had to resort to the complicated Variable Change Lemma (5.3.10). Although this result is intuitive, the proof of the lemma is complicated by the presence of the contraction update. The Conversion Theorem shows that the extension variables behave like real bound variables. Due to this fact, our operational semantics is not sensitive to accidentally used variable names. The informal semantics in [Gur95] lacks this property. There it is assumed that bound and free variables are all different.

The analysis of our semantics yielded two other interesting theorems. The Normal Form Theorem (5.3.19) is used in this thesis twice: firstly, in the proof of the Soundness Theorem (5.3.26), and, secondly, in the Norm-rule of the axiomatic semantics. In fact, this theorem explains how structured update sets can be manipulated. The Isomorphism Theorem (5.3.23) is also used in the proof of the mentioned Soundness Theorem. One of the meanings of this theorem is that the execution of evolving algebras is not sensitive to universe extensions: the new elements can be arbitrarily chosen from the set of available reserve elements.

Finally, we proved soundness and completeness results for our operational semantics. The soundness and completeness of the relation $\xrightarrow{\mathcal{I}}$ implies that our sequential interpretation of parallel updates is independent of the ordering, if the given updates are consistent with respect to $\mathcal{I}$. Completeness ensures that a consistent set can be executed at all. The soundness of the relation $\xrightarrow{\mathcal{R}}$ is, in fact, an extension of the soundness of $\xrightarrow{\mathcal{I}}$ for deterministic evolving algebras which are consistent with respect to $\mathcal{I}$. The completeness of this relation, however, is a weak result: consistent evolving algebras cannot terminate in non-normal final states.

In the last chapter about the axiomatic semantics, logic and evolving algebras come together. The chapter starts with a generalization of the theorems about substitution and the remove operator in Section 4.2. This theorem provides the theoretical background for the Hoare-calculus. It is used in the proof of the soundness of the calculus.

The next theorem relates $\xrightarrow{\mathcal{R}}$ -transitions between states, to $\xrightarrow{\mathcal{R}}$ -transitions between reducts of these states. Then, this theorem is generalized for the relation $\xrightarrow{\mathcal{R}}$. These theorems are useful, since the signature of an
evolving algebra is mostly included in a richer signature for $E$-logic. The richer signature contains auxiliary functions, predicates and constants that enable reasoning about evolving algebras. These theorems ensure that our definition of valid Hoare formulae is adequate. Moreover, the first of these theorems is used in the proof of the soundness of the Hoare-calculus.

The main definition of this chapter contains the actual deduction rules of the Hoare-calculus for proving the partial correctness of evolving algebras. Especially noteworthy is the deduction rule $\texttt{Aux}$ for auxiliary updates. In proving the correctness of an evolving algebra with respect to a given precondition and postcondition, it is sometimes useful, if not necessary, to add auxiliary updates to rules. These updates are used to do additional bookkeeping, while proving the correctness of the enriched evolving algebra. The $\texttt{Aux}$-rule governs the admissible use of these auxiliary updates.

We proved the soundness of this calculus, and show that it is not complete. The incompleteness is probably caused by the contraction update. At least, the proof of the incompleteness uses a single contraction update. The system might be complete without the contraction update. We also introduced a slight modification of the mentioned Hoare-calculus for proving total correctness, and proved its soundness.

The second section of the chapter is devoted to the example of the string reverse evolving algebra. We proved its partial and total correctness. This proof is fairly complicated. The method of proving the correctness is simple, however. The main idea is to find a formula which is an invariant of each rule of the evolving algebra. The invariant for this example is quite involved. One of the reasons for this is the presence of a contraction update. Without it the invariant could be simpler, for in that case we could do without the remove condition $\texttt{Crem}(-,-)$. The proof shows that proving correctness of evolving algebras with dynamic data structures is complicated. The invariant for these kind of algebras has to encode the information which is stored in the structures present in the dynamic universes.

The results of this thesis, as discussed above, show that the original research program, the formalization of many-sorted evolving algebras, can be considered completed. This is not to say that this study is complete, however. In fact, this study can be extended in several ways. We will discuss some of these in the remainder of this chapter.

The first possibility to extend our research program is by adding other evolving algebra constructs, especially those introduced in [Gur95] or the slightly different proposals in [Gur97]. These extensions include, more general rules, external functions, rules with first-order guards, declared variables, nondeterministic choice, and multi-agent evolving algebras.

Since Gurevich makes no distinction between updates and rules, his rules can be more complex than ours. According to his definition, rules can be arbitrarily nested. However, by its relational structure, it is straightforward to adapt our operational semantics to cope with the more general rules of
Gurevich. We only have to make a distinction between nested rules, which appear nested in another rule, and evolving algebra rules, which are the top level rules we considered as rules. Of course, it remains the question whether these more general rules are necessary. At the moment we do not believe so. Too much nesting of rules makes them less perspicuous. However, if the nesting can be removed by rewriting the rule, nesting does not to be harmful per se. For instance, the if ... then ... elseif ... then ... endif proposed in [Gur95] can easily be removed.

External functions are, from the evolving algebra point of view, a sort of oracles. At each invocation of an evolving algebra rule, the environment sets the value of the external functions. This means that external functions can be seen as dynamic functions with a hidden variable, which can only be seen from the outside of the evolving algebra. Coping with external functions can be done by setting this hidden variable at each invocation of a rule at the level of the operational semantics. This implementation would be slightly different from the one in [Gur95], where at each invocation of a rule an external function may be called only once. This ensures that the value of the external function is constant during the execution of the evolving algebra rule.

Rules with first-order guards can straightforwardly be accounted for in our semantics. Of course, if the evolving algebras has to be executable, the domains of quantification have to be finite (in that case the static functions have to be computable as well).

The construct with declared variables makes it possible to execute a rule for each value of a declared variable occurring in that rule. Executing a rule with, what Gurevich calls an explicit atomic variable declaration, can be seen as massive local parallelism, since the same task can be executed by a different processor for a different value of the variable. Clearly, it is intended that the variable ranges over a finite domain.

Rules with nondeterministic choice have variables for which one value is nondeterministically chosen during execution of the rule. This kind of rule seems easy formalizable in our approach.

Evolving algebras with more than one agent are used to specify distributed algorithms. The idea is that a number of agents perform the same task, represented by the same evolving algebra. In order to be able to distinguish the agents, a special universe Agent is introduced, the elements of which are used to identify the agents. In the distributed evolving algebra a special constant Self is used, which for each agent is equal to the value in Agent representing this agent. Perhaps, this feature can be formalized in our semantics by adding a third relation, representing moves of agents.

The task of adapting our axiomatic semantics for these extensions to evolving algebras is not trivial. For example, for more general rules it can be done, if some normal form can be defined for these rules. For other constructs, a lot work has to be done probably. It is not unlikely that the work done in the field of axiomatic semantics for parallel and distributed programming languages (see e.g. [AO91b]) can be of use.
Furthermore, regarding our axiomatic proof system, one could ask the question of what happens if it is being restricted to evolving algebras without remove updates. Is it complete in some sense, in that case?

The second direction to extend the work in this thesis, is to study the proof theory of evolving algebras from a pragmatic point of view. Inspecting the correctness proof of our string reverse algebra, the reader easily is overwhelmed by the complexity of it. To become really useful in practice, proof assistants are necessary. A proof assistant could help to automatically generate the consistency conditions, the remove conditions and the extend conditions, for example. Moreover, the proof assistant could help in verifying simple deductions. Probably, the Fitch-style inference system is not of very much use in that case. Rather, a resolution-based prover should be developed and implemented to achieve this goal.

A third topic for further study has to do with evolving algebra interpreters and pragmatics. The evolving algebra interpreter described in [Vis96] can be extended to cope with the extensions of evolving algebras we discussed above. The current interpreter does not accept evolving algebras with remove updates, however. This feature has still to be implemented. A special feature of the implementation concerns modules. Modules make it possible to describe evolving algebras on different levels of abstraction. This is done by implementing the static functions of a higher level description by evolving algebras encapsulated by a module. The operational and axiomatic semantics of this extension should be studied as well.
Chapter 7. Conclusions and future work
References


[Bör90b] E. Börger. A logical operational semantics of full Prolog; Part II: Built-in predicates for database manipulations. In B. Rovan,
References


References


Samenvatting

Een Theorie van Veelsoortige Evoluerende Algebra’s

Informatici proberen met hun informatieverwerkende systemen een deel van de wereld om ons heen te modelleren. Mensen zijn gewend in deze wereld verschillende typen objecten te onderscheiden. Voorbeelden van verschillende typen zijn gironummers en saldi. In de informatica zijn de typen vaak van meer abstracte aard. Informatici spreken liever over ‘datatypen’. Logici hebben in de loop van de tijd redeneersystemen (logica’s) bedacht om binnen één systeem over objecten van verschillende typen te kunnen redeneren. Veelsoortige logica’s zijn voorbeelden van dergelijke redeneersystemen. ‘Veelsoortig’ slaat in dit verband op het beschikken over verschillende typen, die ook wel ‘soorten’ worden genoemd.

Het ontwerpen en bouwen van een informatieverwerkend systeem is een complexe taak. Architecten en civiele ingenieurs leggen hun ontwerpen vast in bestektekeningen die aan bepaalde conventies voldoen, zodat de bouwers precies weten wat er van hen wordt verlangd. In de informatica worden op analoge wijze talen ontwikkeld waarmee een ontwerp van een informatieverwerkend systeem op ondubbelzinnige wijze kan worden vastgelegd. Dergelijke talen noemt men ‘specificatietalen’. Over het algemeen zijn deze talen op logica of wiskunde gebaseerd, en zijn specificaties van informatieverwerkende systemen in zo’n specificatietaal door hun hoge abstractiegraad vaak moeilijk te begrijpen.

Als reactie hierop heeft de logicus Yuri Gurevich rond 1988 een nieuwe specificatietaal voorgesteld die hij ‘evolving algebras’ (evoluerende algebras) noemde\(^1\). Deze taal is vergeleken met de bestaande specificatietalen erg eenvoudig. Een evoluerende algebra bestaat uit een aantal regels die elk een toestandsverandering kunnen bewerkstelligen. Hierbij verstaat men onder een toestand een abstracte beschrijving van dat (zeer kleine) deel van de wereld dat relevant is voor het informatieverwerkende systeem. Een aardig voorbeeld is de giroboekhouding. Bij elk gironummer hoort een saldo. Het totaal van alle gironummers met de bijbehorende saldi kan men een toestand noemen. Een toestandsverandering treedt bijvoorbeeld op indien de salarissen worden gestort. Dan moeten namelijk de saldi worden aangepast.

Een specificatie die opgeschreven is als een evoluerende algebra is, zoals reeds opgemerkt, een verzameling van regels die ieder een toestandsverandering tot stand kunnen brengen. De betekenis van een dergelijke specificatie

---

\(^1\)Tegenwoordig worden deze ook ‘abstract state machines’ (abstracte toestandsmachines) genoemd.
is dat zij vastlegt hoe een probleem kan worden opgelost in een reeks toestandsveranderingen. Anders gezegd, een evoluierende algebra laat toestanden ‘evolueren’ tot nieuwe toestanden. Iedere toestandsverandering is daarbij het gevolg van het toepassen van één van de regels. Een evoluierende algebra voor de giroboekhouding bevat bijvoorbeeld regels die bepalen hoe bijboekingen of afboekingen kunnen worden gedaan. Of, hoe op bepaalde data de rente voor het eventueel rood staan moet worden uitgerekend.

De versie van evoluierende algebra’s die door Gurevich wordt voorgesteld, is niet veelsoortig. Zijn versie beschikt over slechts één datatype, het ‘superuniversum’ genoemd. Door middel van een voor logici bekende truc kunnen uit dit ene type andere datatypen worden afgeleid (met behulp van zogenaamde ‘karakteristieke functies’). De eerste doelstelling van dit proefschrift was dan ook een veelsoortige versie van evoluierende algebra’s te ontwikkelen zodat men zonder het toepassen van deze truc over verschillende datatypen zou kunnen beschikken.

Vervolgens was het tweede doel om voor deze veelsoortige evoluierende algebra’s een formele semantiek, ofwel een wiskundige beschrijving van hun betekenis, te definiëren. Immers, voor evoluierende algebra’s bestond nog geen bevredigende formele semantiek. In het proefschrift wordt een zogenaamde ‘operationele semantiek’ beschreven. Men zou deze kunnen opvatten als de beschrijving van een machine die regels van evoluierende algebra’s kan uitvoeren. Deze machine is op een wiskundige wijze beschreven in de vorm van een zogenaamde ‘gestructureerde operationele semantiek’.

De derde doelstelling, ten slotte, was het bedenken van een bewijssysteem. Indien een beschrijving van een informatieverwerkend systeem is gegeven in de vorm van een evoluierende algebra, zou men willen weten of die algebra het probleem goed weergeeft. Met behulp van een bewijssysteem kan men bewijzen of de bedachte evoluierende algebra voldoet aan haar specificaties. Deze specificaties worden op hun beurt beschreven met behulp van logische formules die het zogenaamde invoer-uitvoer-gedrag van de evoluierende algebra weergeven. Een eenvoudig voorbeeld: voor het informatiesysteem van de giro moet gelden dat iedere dag alle mutaties worden doorgevoerd. In dit geval bestaat de invoer uit de combinatie van de te verwerken mutatie-opdrachten en alle gironummers met de oude saldi, en de uitvoer bestaat uit alle gironummers met de nieuwe saldi. Om het invoer-uitvoer-gedrag in de vorm van logische formules te kunnen vastleggen, is in het proefschrift een speciale logica beschreven, namelijk ‘veelsoortige E-logica’.

Het proefschrift zit als volgt in elkaar. Hoofdstuk 1 is een algemene inleiding en samenvatting. In hoofdstuk 2 wordt de vraagstelling van het proefschrift nader uitgewerkt en worden de lezer de eerste beginselen van evoluierende algebra’s bijgebracht. Daarna komt in hoofdstuk 3 de grammatica (logici sprekend van de ‘syntaxis’) van veelsoortige E-logica aan bod. Voor deze logica wordt een zogenaamde ‘natuurlijk deductiesysteem’ beschreven. Dit kan men opvatten als een verzameling van regels om te rekenen met formules uit de E-logica. De semantiek van deze formules, ofwel de wiskundige beschrijving van hun betekenis,
wordt in hoofdstuk 4 behandeld. Er wordt bewezen dat het deductiesysteem correct en volledig is. Dat wil zeggen, dat met behulp van de rekenregels van het deductiesysteem precies alle geldige formules kunnen worden verkregen.

In hoofdstuk 5 begint het werk aan evoluerende algebra's. Nadat de syntaxis aan bod is gekomen, wordt de operationele semantiek uitgewerkt. Hierna volgt een groot aantal resultaten dat laat zien dat de semantiek aan alle eisen voldoet die men eraan zou willen stellen. Een belangrijke eigenschap van de semantiek is dat zij correct en volledig is. Dit betekent dat als een evoluerende algebra aan een aantal eisen voldoet, de operationele semantiek (de machine) altijd een welbepaald resultaat oplevert, en dat de semantiek niet ergens tijdens de berekening zomaar ermee ophoudt.

Het bewijsysteem voor evoluerende algebra's wordt in hoofdstuk 6 behandeld. Dit bewijsysteem kan, net als het systeem voor natuurlijke deductie voor E-logica, worden opgetrokken als een stelsel van rekenregels waarmee correctheidsuitspraken kunnen worden verkregen voor evoluerende algebra's. Er wordt bewezen dat dit systeem correct, maar niet volledig is. Correctheid betekent dat de rekenregels geen foute resultaten kunnen opleveren. Dat het systeem onvolledig is, wil zeggen dat niet alle geldige correctheidsuitspraken kunnen worden bewezen. Aan het eind van het hoofdstuk wordt een voorbeeld gegeven van een correctheidsbewijs.

Hoofdstuk 7, ten slotte, geeft een overzicht van de resultaten van het onderzoek zoals beschreven in het proefschrift. Ook wordt daar een aantal aanbevelingen gedaan voor verder onderzoek. De evoluerende algebra's in dit proefschrift kunnen bijvoorbeeld nog worden uitgebreid met een aantal constructies die door Gurevich en anderen zijn bedacht. Voor deze nieuwe constructies moet dan een veelsoortige variant van de syntaxis, de semantiek en het bewijsysteem worden bedacht. Het correctheidsbewijs aan het einde van hoofdstuk 6 is behoorlijk ingewikkeld. Om het vinden van dergelijke bewijzen eenvoudiger te maken zouden computerprogramma's, zogenaamde 'bewijsassistenten', kunnen worden ontwikkeld die de gebruiker daarbij helpen. Uit het bovenstaande volgt dat er nog genoeg is te onderzoeken aan evoluerende algebra's.
Samenvatting
Curriculum vitae


Na een jaar als docent verbonden te zijn geweest aan de HTS-afdeling van de Hogeschool Haarlem, werd hij in 1987 universitair docent bij de sectie Theoretische Informatica van de Faculteit der Technische Wiskunde en Informatica aan de TU Delft. Gedurende de eerste vier jaren was hij in deeltijd gedetacheerd bij de sectie Filosofie van de Faculteit der Wijsbegeerte en Technische Maatschappijwetenschappen. Sedert juni 1994 is hij in deeltijd gedetacheerd als beleidsmedewerker bij de Advanced School for Computing and Imaging.