Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2011,15
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

A Computer-Assisted Proof of the Bellman-Ford Lemma

Peter H. Schmitt

KIT, Karlsruhe Institute of Technology

March 31, 2011

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Fakultät für **Informatik**

Abstract

These notes serve (at least) two purposes. First, they document a proof done with the KeY system of a purely mathematical statement, Lemma 1 below, within the context of Dijkstra's Shortest Path Algorithm. This is an unusual application of the KeY system that is designed to verify Java programs. The verification of a Java implementation of Dijkstra's algorithm itself is the topic of the Diploma thesis [10].

Secondly, we use this simple proof exercise to review the widely practiced method to handle partial functions via underspecification that is also used in the KeY system. Little can be found in the literature on the theoretical foundations of this approach. This report proposes a first step towards a theory of underspecification. Particular emphasis is devoted to the axiomatisation of abstract data types with partial functions.

As a side issue we also include some comments on conservative extensions.

Contents

Chapter 1

A Computer-Assisted Proof

1.1 Basic Definitions

Definition 1 (The Theory of Weighted Graph)

The theory of weighted graphs is formulated with the vocabulary

- **sorts** *node and int*
- **predicate** $edge\ (node, node)$
- **function** $int\ w(node, node)$

and the following axioms

$$\forall node\ n; (!edge(n, n)) \tag{1.1}$$

$$\forall node\ m; (\forall node\ n; (edge(n, m) - > w(n, m) > 0)) \tag{1.2}$$

For formulas we use the asci syntax of the KeY input files even if it looks a bit funny at times. In particular ! denotes negation.

Definition 2 (Weighted Graph)

A weighted graph is any structure satisfying the axioms of the theory of weighted graphs from Definition 1. Thus it is a structure $\mathcal{G} = (V, E, w)$ *with*

1. *V a nonempty set, called the set of* nodes *of the graph*
   This is the interpretation of the sort node. The interpretation of the sort int is not mentioned at this level, since it is fixed.

2. *$E \subseteq V \times V$ a set of pairs of nodes, called edges*
   Thus E is the interpretation of the binary predicate edge.

3. *$w : E \to \mathbb{Z}$ a function that associates an integer with every edge.*
   Here $\mathbb{Z}$ is the interpretation of sort int.

such that the two axioms are satisfied, i.e.;

1. *$\forall v \in V((v, v) \notin E$, i.e., a graph contains no* self-loops.

2. *$\forall (n, m) \in E(w(n, m) > 0)$ i.e., there are no edges with negative or zero weight.*

In other contexts weights are real-valued, i.e., take values in the set $\mathbb{R}$ of real numbers. For the purposes of the proof of Lemma 1 below, this does not make any difference.

There is another issue involved here. Strictly speaking, the weight function w is a partial function, we only need weight for nodes that are connected by an edge. In the KeY approach partial functions are handled by the principle of underspecification. Thus w is always considered as a total function. Since no particular assumptions are made about the values of w for non-connected pairs of nodes this has the same effect as partiality. We will come back to this issue later.

Definition 3 (Path)

A path s in a weighted graph $\mathcal{G} = (V, E, w)$ is a (finite) sequence $v_0, \ldots, v_k$ of nodes such that for all $0 \leq i < k$ the pair (v_i, v_{i+1}) is an edge of $\mathcal{G}$, i.e., $(v_i, v_{i+1}) \in E$.
In particular every sequence $s = v_0$ of length 1 is a path.

We extend our initial theory by adding a unary predicate fwpath(Seq) and the defining axiom

$$
\begin{aligned}
&\forall Seq\ seq; (\text{fwpath}(seq)\ <->\\
&\ \forall int\ iv; ((0 <= iv\ \&\ iv < seqLen(seq) - 1) ->\\
&\quad edge(node :: seqGet(seq, iv), node :: seqGet(seq, iv + 1))\ \&\\
&\quad node :: instance(any : seqGet(seq, iv))\ \&\\
&\quad node :: instance(any : seqGet(seq, seqLen(seq) - 1)))
\end{aligned}
\tag{1.3}
$$

We make use of the abstract data type Seq built into the KeY system. The function $seqLen(seq)$ obviously denotes the length of the sequence seq while $node :: seqGet(seq, iv)$ denotes the element at position iv in the sequence seq cast to type $node$. The first position is 0. The issue of underspecification of the partial function $node :: seqGet(seq, iv)$ also arises here. Since we do not use polymorphic types the element $any :: seqGet(seq, iv)$ stored in the sequence seq at position iv could be of arbitrary type, any is the universal type that is a supertype to all types. The function $node :: seqGet(seq, iv)$ is undefined if $any :: seqGet(seq, iv)$ is not of type $node$. As part of the defining axiom (1.3) we require that all elements stored in a sequence seq that satisfies fwpath(seq) are of sort $node$.

The name of the predicate is accidentally named fwpath since in an early version there was also a related notion called $fdpath$.

It is convenient for doing proofs with the KeY system to add for axioms defining new functions or predicates as e.g., axiom 1.3 a proof rule that unfolds the definition. In the case of axiom 1.3 unfolding, i.e., the application of the $\rightarrow$-part of the equivalence, suffices. It will turn out that the reverse implication is not needed. For the sake of the experts the rule is presented as a taclet in Section 4.

Among the functions defined for the abstract data type of sequences is the concatenation function $seqConcat$ defined by

$$
seqConcat(s_1, s_2)[i] = \begin{cases} s_1[i] & \text{if } i < seqLen(s_1) \\ s_2[i - seqLen(s_1)] & \text{if } i \geq seqLen(s_1) \end{cases}
$$

For brevity we wrote $s[i]$ instead of $seqGet(s, i)$ and suppressed casting. The singleton sequence $\langle a \rangle$ with sole element a is denoted by $seqSingleton(a)$.

Proofs of formulas of the form $\forall seq\ s; b(s)$ will most of the time be done by some kind of induction. Induction on the length of s would be one possibility. More convenient is the structural induction rule

$$
\frac{b(\langle\rangle) \quad \forall objx; (b(s) \rightarrow b(s + \langle x \rangle))}{\forall seq\ s; b(s)}
$$

Here, $s + \langle x \rangle$ is shorthand for $seqConcat(s, seqSingleton(x))$.

It is occasionally handy to also have the following variant of the structural induction rule available:

$$\frac{b(\langle\rangle) \quad \forall obj x; (b(s) \rightarrow b(\langle x \rangle) + s)}{\forall seq\ s; b(s)}$$

Here, $\langle x \rangle + s$ is shorthand for $seqConcat(seqSingleton(x), s)$.

We have presented the induction rules in mathemantical style, the taclet formalisations may be found in Subsection 4.1. Most of the time when proving a formula $\forall seq\ s; b(s)$ the induction rule is used to first prove $\forall seq\ s; a(s)$ and then to show that $\forall seq\ s; a(s) \rightarrow \forall seq\ s; b(s)$ is universally valid. This is the reason why the taclets for the induction rules break done one proof goal into three subgoals.

Definition 4 (Weight of a Path)

The weight of a path $s = v_0, \ldots, v_k$, denoted by $pw(s)$, is the sum of the weights of its edges, i.e., $pw(s) = \Sigma_{i=0}^{i=k-1} w((v_i, v_{i+1}))$.
For the degenerate case $k = 0$, a path of length 1, we define $w(s) = 0$.

It has become customary to call a path $s = v_0, \ldots, v_k$ a shortest path from v_0 to v_k if for any other path $t = x_0, \ldots, x_n$ with $x_0 = v_0$ and $x_n = v_k$ we get $pw(s) \leq pw(t)$. This terminology may be a bit confusing, since it is not the number of nodes in a path that count. To be consistent we should speak of a path with least weight. But, you cannot beat common practise. In particular, the one element path v is the shortest path from v to v.

To formalize the pathweight function we make use of the *bounded sum* function *bsum* available with the KeY system. The syntax is

$$bsum\{iv\}(t_1, t_2, t)$$

where

1. iv is a variable of type *int*
2. t_1, t_2 are terms of type *int* that do not contain iv
3. t is a term of type *int* that typically will, but need not, contain the variabel iv.

The term $bsum\{iv\}(t_1, t_2, t)$ is evaluated in a structure $\mathcal{A}$ and variable assignment β as

$$bsum\{iv\}(t_1, t_2, t)^{(\mathcal{A},\beta)} = \begin{cases} 0 & \text{if } b \leq a \\ \Sigma_{i=a}^{i<b}\ c(i) & \text{otherwise} \end{cases}$$

with

1. $a = t_1^{(\mathcal{A},\beta)}$
2. $b = t_2^{(\mathcal{A},\beta)}$
3. $c(i) = t^{(\mathcal{A},\beta_i)}$ with $\beta_i(v) = \begin{cases} i & \text{if } v \equiv iv \\ \beta(v) & \text{otherwise} \end{cases}$

Definition and lemmas for *bsum* are shown in Subsection 4.3. For the cases where no obvious value exists, i.e., when the upper bound is less or equal to the lower bound, one option would have been to leave the sum value undefined. Here another decision was taken: the value is 0, see taclet `bsum_empty` in Subsection 4.3.

Using the *bsum* function provided by the KeY system we define the weight of a path by:

$$\begin{aligned} &\forall Seq\ seq; (\\ &pw(seq) \quad = \quad bsum\{iv\}(0, seqLen(seq) - 1, \\ &\qquad\qquad w(node :: seqGet(seq, iv), node :: seqGet(seq, iv + 1)))) \end{aligned} \tag{1.4}$$

A taclet for unfolding Definition 1.4 is again shown in Subsection 4.1.

Definition 1.4 entails the following special cases:

$$\begin{aligned} pw(\langle\rangle) &= 0 \\ pw(\langle a\rangle) &= 0 \\ pw(\langle a, b\rangle) &= w(a, b) \end{aligned}$$

1.2 The Bellman-Ford Lemma

We want to prove the following lemma.

Lemma 1 (Bellman-Ford Lemma)
Let $\mathcal{G} = (V, E, w)$ a weighted graph, start $\in V$. Let $d : V \to \mathbb{N}$ be a mapping satisfying the following properties

1. $d(start) = 0$

2. $\forall m \in V \setminus \{start\} \exists n \in V((n, m) \in E \wedge d(m) = d(n) + w(n, m))$

3. $\forall m \forall n((n, m) \in E \rightarrow d(m) \leq d(n) + w(n, m))$

Then $d(n)$ is the weight of the shortest path from $start$ to n in $\mathcal{G}$.

The properties 2 and 3 are called the *Bellman-Ford Equations*.
They are sometimes summarized as
$\forall\ m \in V \setminus \{start\}(d(m) = min\{d(n) + w(n, m) \mid (n, m) \in E\}$

The assumptions of the lemma are formalized as follows.

$$\forall node\ n; (d(n) >= 0) \tag{1.5}$$

This assumption is somewhat hidden in that the range of d is given as $\mathbb{N}$ and not as $\mathbb{Z}$.

$$d(start) = 0) \tag{1.6}$$

$$\begin{array}{l} \forall node\ m; (m! = start- > \\ \exists node\ n; (edge(n, m)\ \&\ d(m) = d(n) + w(n, m))) \end{array} \tag{1.7}$$

First Bellman-Ford equation.

$$\forall node\ m; (\forall node\ n; (edge(n, m)- > d(m) <= d(n) + w(n, m))) \tag{1.8}$$

Second Bellman-Ford equation.

The claim of the lemma is split into the following two claims:

$$\begin{array}{lll} (\forall node\ m; (m! = start & - > & (\exists Seq\ s; (\text{fwpath}(s)\ \& \\ & & node :: seqGet(s, 0) = start\ \& \\ & & node :: seqGet(s, seqLen(s) - 1) = m\ \& \\ & & pw(s) = d(m)))) \end{array} \tag{1.9}$$

$$\begin{array}{lll} \forall node\ m; (\forall Seq\ s; & & ((\text{fwpath}(s)\ \& \\ & & node :: seqGet(s, 0) = start\ \& \\ & & node :: seqGet(s, seqLen(s) - 1) = m) \\ & - > & d(m) <= pw(s)))) \end{array} \tag{1.10}$$

We observe that part of the claim, formula 1.9, is that the Bellman-Ford equations also entail that every node in the graph is reachable from $start$. This restriction could be waived by considering $d : Node \rightarrow \mathbb{N} \cup \{\infty\}$. We will not persue this here, at least not on the first attempt.

1.3 Auxiliary Concept

The second Bellman-Ford equation guarantees that for all edges (v_i, v_{i+1}) the inequality $d(v_{i+1}) \leq d(v_i) + w(v_i, v_{i+1})$ is true. It will turn out that paths such that all its edges satisfy the stricter requirement of equality will play a prominent role in the proof. In the following definition we give these special paths a name.

Definition 5
A path $s = v_0, \ldots, v_k$ *in a weighted graph* $\mathcal{G} = (V, E, w)$ *is called* faithful with respect to d *or for short a* d-path *if for all* $0 \leq i < k$

$$d(v_{i+1}) = d(v_i) + w(v_i, v_{i+1})$$

As a formula this definition reads as:

$$\begin{array}{lll} \forall Seqs; (dpath(s) & \leftrightarrow & \text{fwpath}(s)\ \& \\ & & \forall int\ i; (0 <= i \& i < seqLen(s) - 1 - > \\ & & d(s[i+1]) = d(s[i]) + w(s[i], s[i+1])) \end{array} \tag{1.11}$$

A taclet for unfolding Definition 1.11 is again shown in Subsection 4.1.

1.4 Proof

We are faced with showing universal validity of the following the implications:

$$\begin{array}{lll} (1.1)\ \&\ (1.2)\ \&(1.3)\ \&\ (1.4)\ \&(1.5)\ \& & & \\ (1.6)\ \&\ (1.7)\ \&\ (1.8)\ \&\ (1.11) & \rightarrow & (1.9) \\ (1.1)\ \&\ (1.2)\ \&(1.3)\ \&\ (1.4)\ \&(1.5)\ \& & & \\ (1.6)\ \&\ (1.7)\ \&\ (1.8)\ \&\ (1.11) & \rightarrow & (1.10) \end{array}$$

In the KeY system these goals will be written as sequents:

$$\begin{array}{lll} (1.1)\ \&\ (1.2)\ \&(1.5)\ \&\ (1.6)\ \&\ (1.7)\ \&\ (1.8) & ==> & (1.9) \\ (1.1)\ \&\ (1.2)\ \&(1.5)\ \&\ (1.6)\ \&\ (1.7)\ \&\ (1.8) & ==> & (1.10) \end{array}$$

The definitions (1.3), (1.11), (1.4) are available as taclets and need not be stated as assumptions in the antecedent.

Halfway through the proof I noticed that (1.2) & (1.8) $\rightarrow$ (1.1) i.e., the axiom (1.1) is redundant. In the description of the various proof subgoals axiom (1.1) is still listed but it has never been used.

In a desperate attempt one could ask the system to prove these goals without any further help or interaction. As expected this does not work, after 5 000 steps 229 goals have been generated and not a single one has been closed.

We will describe in detail the guidance and interactions necessary to produce a computer assisted proof with the KeY system. All proof obligations are available in electronic form as input files for the KeY- system. Likewise, the files that store the completed proofs.

Claim	input file name	proof file name
(1.12)	Lemma1.key	Lemma1.key.proof
(1.13)	Lemma2.key	Lemma2.key.proof
(1.14)	Lemma3.key	Lemma3.key.proof
(1.15)	Lemma4.key	Lemma4.key.proof
(1.16)	Lemma5.key	Lemma5.key.proof
(1.10)	seqBFTheoremPart2.key	seqBFTheoremPart2.key.proof

1.4.1 Proof of Claim 1

We propose to derive the following series of lemmata (1.12) through (1.16). We will first present the lemma to be proved followed by some comments on the interactive proof itself.

$$\begin{aligned}&\forall Seq\ s; (fdpath(s) - > \\ &d(node :: seqGet(s, seqLen(s) - 1)) >= seqLen(s) - 1 | seqLen(s) = 0)\end{aligned} \tag{1.12}$$

This is an easy observation about d-paths. If $s_0, \ldots s_{n-1}$ is a d-path then $d(s_{n-1}) \geq n$. By the definition of a d-path we have $d(s_{n-1}) = d(s_0) + \Sigma_{i=0}^{i=n-2} w(s_i, s_{i+1})$. Using the facts $d(s_0) \geq 0$ and that all weights are strictly positive a human reasoner will immediately believe the lemma. Of course attention needs to be paid to the borderline cases of an empty or one-element sequence. For a formal proof we need to use structural induction on sequences. KeY finds a proof with 4227 nodes, 13 branches and 19 interactive steps. The first interaction is to start structural induction, which is the obvious choice since the claim is of the form $\forall Seqs; (\ldots)$. The initial and use case of the inductive proof complete automatically. The induction step is split into two top cases. First, we need to show that when $s + \langle a \rangle$ is a d-path then also s is an d-path. The second part is the derivation of the inductive claim from the induction hypothesis. In both cases it suffices to unfold the definitions of d-path and fw-path. KeY does the rest automatically. It is notable that instantiation of the universal quantifiers in (1.2) and (1.5) are found automatically.

Not all assumptions are used in this proof. In fact it is shown that (1.2) &(1.3) & (1.5) & (1.11) → (1.12) is true.

$$\begin{aligned}&\forall int\ i; (i >= 0- > \forall node\ m; (\exists Seq\ s; (\\&\quad (fdpath(s)\ \&\\&\quad node :: seqGet(s, seqLen(s) - 1) = m\ \&\\&\quad seqLen(s) >= 1\ \&\\&\quad (node :: seqGet(s, 0) = start | seqLen(s) >= i))))\end{aligned} \tag{1.13}$$

To understand the motivation behind this lemma we observe that in order to satisfy claim 1 (formula with number (1.9)) we need for any node n an d-path that ends in n and begins in $start$. The first Bellman-Ford equation (1.8) allows us to find for any d-path that ends in n and begins with m to find a longer path that ends in n and now begins with a predecessor m_1 of m. If we are lucky we will hit in this way upon $start$ as a first node in the path. Otherwise the path gets longer and longer. That is what we prove here. In Lemma 3 (formula with number (1.14)) below we exploit the fact that by Lemma 1.12 the length of a d-path ending in m cannot grow beyond $d(m)$. So we are sure to hit $start$ eventually.

We now step through the interactions need for a KeY proof of formula (1.13).

The first interaction consists in starting integer induction. Strengthening of the formula to be proved is not necessary.

For the **initial case**, $i = 0$, the formula $\forall node\ m; (\exists Seq\ s; (\ldots)$ is proved by instantiating s by $seqSingleton(m\_0)$ where m_0 is the Skolem constant for $\forall node\ m$. After manually triggering the unfolding of $fdpath$ and fwpath the proof completes automatically.

The **use case** is trivially closed in one step.

The main work is with the **step case**. It schematically is of the form $\forall node\ m; \exists seq\ s; (\phi(m, s, i) \Rightarrow \forall node\ m; \exists seq\ s; (\phi(m, s, i + 1))$. Let m_1 be the Skolem constant for the righhand side $\forall node\ m$. Instantiate the lefthand side $\forall node\ m$ with m_1 we obtain a Skolem constant s_0 with $\phi(m\_1, s\_0, i)$. At this point it is clever to perform a case destinction whether $node :: seqGet(s_0, 0) = start$ or not. If $node :: seqGet(s_0, 0) = start$ then we can instantiate the quantifier $\exists seq\ s; (\phi(m\_1, s, i + 1))$ with s_0. The

sequence already starts with the *start* node and need not be extended. After manually unfolding all $fdpath$ and fwpath functions KeY completes the proof automatically.
Thus the case $node :: seqGet(s_0, 0)\ !=start$ remains. Instantiating the leading universal quantifier of the first Bellman-Ford equation with $node :: seqGet(s_0, 0)$ we obtain a node n_0 with

$$edge(n_0, node :: seqGet(s_0, 0)) \text{ and}$$
$$d(node :: seqGet(s_0, 0)) = d(n_0) + w(n_0, node :: seqGet(s_0, 0))$$

For this $node :: seqGet(s_0, 0)\ != start$ is necessary. Now we are able to instantiate the existential quantifier $\exists seq\ s; (\phi(m\_1, s, i+1))$ on the righthand side by $seqConcat(seqSingleton(n\_0), s\_0)$ and after the usual unfolding of $fdpath$ and fwpath the proof is completed automatically. In total 610 proof nodes and 8 branches have been generated with 40 interactions.

Very few assumptions are needed. Indeed (1.3) & (1.11) & (1.7) $\rightarrow$ (1.13) is proved.

$$\begin{aligned}
&\forall node\ m; (\exists Seq\ s; (\\
&\quad fdpath(s)\ \& \\
&\quad seqLen(s) >= 1\ \& \\
&\quad node :: seqGet(s, seqLen(s) - 1) = m\ \& \\
&\quad node :: seqGet(s, 0) = start))
\end{aligned} \tag{1.14}$$

As already aluded to in the motivation for Lemma 2 (formula number (1.13)) we propose to prove: (1.2) & (1.12) & (1.13) $\Rightarrow$ (1.14).
Let m_0 be the Skolem constant introduced by eliminating the universal qantifier $\forall node\ m; (\ldots)$ on the righthand side and instantiating $\forall int\ i; (\ldots)$ on the lefthand side with $d(m\_0)+2$ then Z3 can solve the problem immediately (the time shows 0.0sec). KeY on the other hand cannot do this. Only after unfolding all existential quantifiers and instantiating all universal quantifiers, about a handful of interactions, KeY finds the rest of the proof automatically.

$$\begin{aligned}
&\forall Seq\ s; (\\
&(fdpath(s)\ \&\ seqLen(s) >= 2\ \&\ node :: seqGet(s, 0) = start) \\
&- > \\
&d(node :: seqGet(s, seqLen(s) - 1)) = pw(s))
\end{aligned} \tag{1.15}$$

More precisely we show (1.3) & (1.4) & (1.6) & (1.11) $\Rightarrow$ (1.15)

For a human this lemma is obviously true. A formal proof needs some kind of induction:

$$\begin{array}{ll} \forall int\ i; \forall Seq\ s; (& fdpath(s)\& \\ & seqLent(s) = i\& \\ & seqLen(s) >= 1)\& \\ & node :: seqGet(s, 0) = start \\ & - > d(node :: seqGet(s, sub(seqLen(s), 1)))) \\ = pw(s)) & \end{array}$$

The initial case is trivial since $seqLen(s) = 0\&seqLen(s) >= 1$ on the left-hand side of the implication to be proved is contradictory. KeY finds this automatically. After interactive instantiation of $i = seqLen(s\_0)$ where $s\_0)$ is the Skolemconstant for the quantifier $\forall Seq\ s$ the proof completes automatically.

The step case is now the only open goal. Let i_0 be the Skolem constant for the universal integer quantification. We know $i\_0 >= 0$. At this point it makes sense to distinquish the cases $i = 0$ and $i\_0 > 0$. The first case can be closed with a few interactions (did not check automatic completion)

In the step case let s_4 be a Skolem constant for the universal quantification to be proved. We make use of the induction hypothesis for the atomic formula $seqSub(s\_4, 0, i\_0 - 1)$. Before the crucial part of the induction hypothesis can be used, we have to show that

1. $seqSub(s\_4, 0, i\_0)$ is an d-path given that s_4 is a d-path
2. $seqLen(seqSub(s\_4, 0, i\_0)) = i\_0$
3. $seqLen(seqSub(s\_4, 0, i\_0)) >= 1$
4. $node :: seqGet(seqSub(s\_4, 0, i\_0), 0) = start$
   given that $node :: seqGet(s\_4, 0) = start$.

It only takes unfolding of the definitions of *pdpath* and fwpath to prove 1 to 4 automatically. Note, that we need $i\_0! = 0$ for 3. This shows why the case distinction was useful. Now we know

$$\begin{array}{rcl} d(node :: seqGet(s\_4, i\_0)) & = & d(node :: seqGet(s\_4, i\_0 - 1)) + \\ & & w(node :: seqGet(s\_4, i\_0 - 1), node :: seqGet(s\_4, i\_0)) \end{array}$$

plus

$$d(node :: seqGet(s\_4, i\_0 - 1)) = pw(seqSub(s\_4, 0, i\_0 - 1))$$

and need to prove

$$d(node :: seqGet(s\_4, i\_0)) = pw(s\_4)$$

After unfolding pw the proof completes almost automatically. Suprisingly the taclet $bsum\_one\_summand$ has to be triggered interactively.
Statistics 715 nodes, 9 branches, 74 interactions.

Finally we show

$$(1.14) \ \& \ (1.15) \Rightarrow (1.9) \tag{1.16}$$

This only needs the obvious instantiations of universal quantifiers left and existential quantifiers right.

Summary We summarize the proof efforts for the various parts in the proof of claim 1 in the following table.

Task	nodes	branches	inter-actions
(1.2) &(1.3) & (1.5) & (1.11) $\rightarrow$ (1.12)	4227	13	19
(1.3) & (1.11) & (1.7) $\rightarrow$ (1.13)	610	8	40
(1.2) & (1.12) & (1.13) $\Rightarrow$ (1.14)	83	4	24
(1.3) & (1.4) & (1.6) & (1.11) $\Rightarrow$ (1.15)	715	9	74
(1.14)&(1.15) $\Rightarrow$ (1.9)	82	2	11
Total	5697	36	168

The degree of automation thus is 97,1% without using Z3.

1.4.2 Proof of Claim 2

We prove claim 2 (formula with number (1.10)) directly without any intermediate lemmata.

We start, unsurprisingly, by triggering structural induction on sequences, taclet $seqInd\_forward$. No strengthening if the induction hypothesis is necessary. Therefore the use case of the induction immediately is closed. Also the initial step is quickly and automatically handled, since the righthand side of the implication to be proved is obviously false. The leaves us struggling with the step case. Instantiating the two leading universal quantifiers we obtain Skolem constants s_0 and n_0. The principle of the preservation of happiness takes its toll here. The initial step was trivial. In the step case we need a case distinction $s\_0 = seqEmpty$ and $s\_0 \mathrel{!=} seqEmpty$.

If $s\_0 = seqEmpty$ then the claim to be proved reduces to $d(n\_0) <= pw(seqSingleton(n\_0))$. By definition of $bsum$ with equal lower and upper bound we get $pw(seqSingleton(n\_0)) = 0$. It takes to interactions to trigger direct cuts to obtain $n\_0) = start$. KeY finds the assumption $d(start) = 0$ and finishes this case automatically. Note, the induction hypothesis is not used here.

From now on we have $s\_0 \mathrel{!=} seqEmpty$ as an additional assumption. Again we first rewrite the argument of $d()$ in the claim to n_0. One_Step_-Simplification is not powerful enough for this. Twice a direct cut has to be triggered. Fortunately, the rest of the proof complete automatically. The claim to be proved has so far been reduced to

$$d(\_0) <= pw(seqConcat(s\_0, seqSingleton(n\_0))).$$

To prepare the application of the induction hypothesis we unfold the definition of pw and split the resulting sum with the taclet $bsum\_split$:

$$\begin{array}{lll} d(n\_0) & <= & \Sigma_{j=0}^{j=seqLen(s\_0)-2} \\ & & w(node :: seqGet(seqConcat(s\_0, seqSingleton(n\_0)), j), \\ & & \ node :: seqGet(seqConcat(s\_0, seqSingleton(n\_0)), j+1)) \\ & & +w(node :: seqGet(s\_0, seqLen(s\_0) - 1), n\_0) \end{array}$$

After proving

1. fwpath(s_0)

2. $node :: seqGet(s\_0, 0) = start$

3. $seqLen(s\_0) >= 1$

which is done by apppying the rule $implication\_left$ unfolding for the two occurences of fwpath its definition and let KeY do the rest automatically, we have the induction hypothesis $d(node :: seqGet(s\_0, seqLen(s\_0) - 1)) <= pw(s\_0)$ at our disposal.

Now we face a serious problem. We have to show that $bsum\_1 = bsum\_0$ for

$$\begin{array}{lll} bsum\_1 & = & \Sigma_{j=0}^{j=seqLen(s\_0)-2} \\ & & w(node :: seqGet(seqConcat(s\_0, seqSingleton(n\_0)), j), \\ & & \quad node :: seqGet(seqConcat(s\_0, seqSingleton(n\_0)), j+1)) \end{array}$$

$$bsum\_0 = \Sigma_{j=0}^{j=seqLen(s\_0)-2} w(node :: seqGet(s\_0, j), node :: seqGet(s\_0, j+1))$$

It is a formidable task for symbolic deduction to show that two sums $\Sigma_{i=0}^{i=b} t_i$ and $\Sigma_{i=0}^{i=b} s_i$, even with the same bounds, are equal. This would , e.g., be true for any permuation $s_0, \ldots, s_b$ of $t_0, \ldots, t_b$. A human might immediately spot this, but it would be a laborious job to cast this in proof rules. Fortunately, the case at hand is of a simpler nature. While the summands s_i and t_i are syntactically different, it can be proved that they always evaluate to the same element with the given bounds of the summation variable. The taclet *equal_bsum*3 does this. The *pull_out* taclet has to be used twice to prepare for an application of *equal_bsum*3.

Summarizing, we have at this point

$$d(n\_1)) \quad <= \quad bsum\_1 \tag{1.17}$$

with $n\_1 = node :: seqGet(s_0, seqLen(s_0) - 1)$.

Manual instantiations of the leading universal quantifiers in the second BellmanFord equation gives us

$$d(n\_0) \quad <= \quad d(n\_1) + w(n\_1, n\_0) \tag{1.18}$$

After (1.17) and (1.18) are available KeY finds automatically that

$$d(n\_0) \quad <= \quad bsum\_1 + w(n\_1, n\_0)$$

follows. But, this completes the proof of Claim 2.
Statistics 1281 nodes, 17 branches and 92 interactions. Thus we still have a degree of automation of 92.82%.

1.5 Conclusion and Outlook

Given the fact that the KeY system was developed for program verification and not as a mathematical proof assistent I was surprised how well it could be used for exactly this purpose. After a close inspection of the lemma I arrived at a fairly detailed proof plan and the KeY system offered all the possibilities to carry it out without any compromise. Before the final version of the proof as presented in Section 1.4 was found I went through some failed attempts. They all had to do with inductive proofs. Strenghtening the induction hypothesis was not a problem, that was either not needed at all or really easy. It was the borderline cases that caused me trouble. E.g., the empty sequence or one-element sequences required special attention which I did not get right at first go.

I did not try hard to minimise the number of interaction. Besides the interactions needed to implement my proof plan:

1. find a series of lemmata to be proved and in which order
2. trigger induction
   (integer induction or structural induction on sequences)
3. trigger case distinctions

it was instantiations of universal quantifiers (to be utterly precise of universal quantifiers on the left and existential quantifiers on the right of the sequent seperator) and the choice of rules manipulating bounded sums.

I had some qualms about the liberal way partial functions were treated, in the theory of weighted graphs, and in the theories of sequences and bounded sums. I am now quite satisfied with the framework developed in Section 2. Considerable effort is still needed to establish the assumptions of Theorem 8, but we believe that this is due to the inherent complexity of the axiomatisation and cannot be reduced.

Do I have suggestions for improvements of the KeY tool? Yes.

1. Maybe one could add an (automatic) modification of the taclet *bsum _one _summand*

   ```
   bsum_one_summand_auto {
   \find(bsum{uSub;} (i0, i1, i1+1))\sameUpdateLevel
   \varcond (\notFreeIn(uSub, i0),
                        \notFreeIn(uSub, i1))
   \replacewith({\subst uSub; (INT2) i0} t)
                      \heuristics(simplify)
   };
   ```

2. When calling an (or all) SMT solver(s) the user can choose in a dialog if he want to apply or ignore the result. It would be very convenient to also have this feature for the `apply rules automatically here`.
   This would be in particular useful for exploring the limits of interaction.

In a lot of proof situations I tried wether KeY would find the solution automatically from here. If it did, wonderful. If not, I pruned the proof tree back to where I started and did some further interactions. It works, but is laborious.

3. A robustly working save and load mechanism for (partial) proofs is absolutely mandatory. The user of the KeY system should be able to interrupt his work and continue the other day where he left off. Also he may want to exchange completed proofs with others.
   That was a valid remark at the time I did the experiments. Now, loading and saving works. Still, the mechanism is rather fragile.

4. Some kind of replay mechanism of a (complete or partial) proof in a slightly different setting would be highly welcome.
   E.g., one would like to run a completed proof again without one of the assumptions to find out if that assumption was really used? I do not mean to find out if a proof without this assumption exists, that might be a difficult mathematical task, only if in the given proof it was used. Another situation arises when the definition of a function in an abstract datatype specification is slightly changed. The crucial information is, when to unwind the definition, the rest should work automatically.
   There is no limit to ambitions. I know, that is no easy task. I suggest we commit ourselves to facing it.

A desirable simple extension of Lemma 1 would be to drop the restriction that every node in the graph should be reachable from the start node. One way to achieve this would be to change the typing of the distance estimation function to $d : V \to \mathbb{N} \cup \{\infty\}$ with the Bellman-Ford equations accordingly modified:

$$\begin{aligned}&\forall node\ m; (m! = start\ \&\ d(m) \neq \infty \to \\ &\quad \exists node\ n; (edge(n,m)\ \&\ d(m) = d(n) + w(n,m)))\end{aligned} \tag{1.19}$$

$$\begin{aligned}&\forall node\ m; (\forall node\ n; (edge(n,m)\ \&\ d(n) \neq \infty) \to \\ &\qquad\qquad d(m) <= d(n) + w(n,m)))\end{aligned} \tag{1.20}$$

It is however not so easy to replace the type $\mathbb{N}$ by $\mathbb{N} \cup \{\infty\}$, that type theoretically is the disjunction of type $\mathbb{N}$ and the singleton type $\{\infty\}$. Also, in the course of Dijkstra's algorithm $d(m) = \infty$ signifies that $d(m)$ has not been defined yet. This suggests to model d as a partial function. The fixed value formula is $fix_{d(x)} = reach_{start}(x)$ where $reach_{start}(node)$ is a new atomic predicate. With this axiomatisation the Bellman-Ford equations take on the form:

$$\begin{aligned}&\forall node\ m; (m! = start\ \&\ reach_{start}(m) \rightarrow \\ &\ \exists node\ n; (edge(n, m)\ \&\ d(m) = d(n) + w(n, m)))\end{aligned} \tag{1.21}$$

$$\begin{aligned}&\forall node\ m; (\forall node\ n; (edge(n, m)\ \&\ reach_{start}(n) \rightarrow \\ &\qquad\qquad\qquad d(m) <= d(n) + w(n, m)))\end{aligned} \tag{1.22}$$

The claim of the lemma corresponding to Lemma 1 now read:

$$\begin{aligned}\forall node\ m; (m! = start\ \ &\&\ reach_{start}(m)\\ &-> \ (\exists Seq\ s; (\text{fwpath}(s)\ \&\\ &\qquad node :: seqGet(s, 0) = start\ \&\\ &\qquad node :: seqGet(s, seqLen(s) - 1) = m\ \&\\ &\qquad pw(s) = d(m))))\end{aligned} \tag{1.23}$$

$$\begin{aligned}\forall node\ m; (\forall Seq\ s; \qquad &((\text{fwpath}(s)\ \&\\ &node :: seqGet(s, 0) = start\ \&\\ &node :: seqGet(s, seqLen(s) - 1) = m)\\ -> \ &d(m) <= pw(s)\ \&\ reach_{start}(m)))\end{aligned} \tag{1.24}$$

From 1.23 and 1.24 it follows in particular that $reach_{start}(m)$ is true if and only if m is reachable from $start$.

Chapter 2

Partial Functions

2.1 Motivation and Introduction

While in classical mathematical logic partial functions were at best addressed as a side issue for logic in computer science geared towards applications it is absolutly essential to deal with them transparently and efficiently. Different specification and verification languages, however, offer different solutions. The situation has been carefully investigated in [9]. The author argues, convincingly, that many-valued logic is not well suited for the purposes of specification and supports the approach called *underspecification*. This approach can be traced back to the paper [8] that also contains references to earlier research.

The way the KeY system implements underspecification is concisely explained in [4, page 90]. A greater part of this explanation is concerned with Kripke structures with partial observer functions as needed for the semantics of Dynamic Logic. Also the papers [15, 16], [7] and [14] focus on the treatment of partial functions in software specifications and program annotations. Here we will be mainly concerned with partiality in data types.

While [8] only gave an informal explanation of underspecification the reference [9, Section 5.1] includes a formal definition. Our presentation below improves on that by also paying attention to the specification of the domain of partial functions and by formalizes the notion of logical inference.

The two main advantages of the method of underspecification is that classical first-order logic can be used for reasoning without any changes and the need to define evaluation of terms and formulas in partial structures is sidestepped. But, what are the drawbacks? The main nuisance is the fact that a lot of unintuitive or even strange properties can be derived.

It is a simpler exercise to understand that $1/0 \doteq 1/0$ is universally valid, while $1/0 \doteq 2/0$ is not.It may also not be too hard to accept the statement $\forall seqs\ \exists seq\ t(t \doteq seqSub(s, -1, -5))$. But, would you have expected that

$$\exists seq\ s(seqLen(s) \doteq 0\ \&\ int :: seqGet(seqSub(s, 1, 2)) \doteq 5)$$

can be proved from the proof rules in Subsection 4.2?

These example formulas share a common feature: they are in some sense not well defined. The solution taken by almost all verification systems is to construct for every formula ϕ a well-definedness condition wd_ϕ. The intention is that if $T \vdash wd_\phi$ can be derived then ϕ is well-defined in all models of the underlying theory T. The proof task $T \vdash \phi$ is then augmented to $T \vdash \phi \wedge wd_\phi$.

The seemingly innocent question *When is a formula well-defined?* does not have a straight forward answer. There are many plausible answers. In the JML manual [12] well-definedness is part of the notion of validity:

> An assertion is taken to be valid if and only if its interpretation
>
> - does not cause an exception to be raised, and
> - yields the value true.

An attempt to cast this into a comprehensive formal semantics is contained in the Diploma Thesis [6].

We could try to adapt this to our fist-order logic context and replace what in this quote is called *interpretation* by the usual recursive definition of truth of a formula in a given structure. If during this recursive evaluation we need to evaluate a function on arguments outside its domain of definition we could view this as the analogon of *exceptions* refered to in the JML context. If during the recursive evaluation we never need to evaluate a function on arguments outside its domain of definition then we declare the formula to be well-defined. The problem with this approach is that it crucially depends on the way the truth value of a formula is evaluated. If we encounter during the computation of the truth value of a conjunction $\phi \wedge \psi$ the situation that ϕ is false but ψ may be undefined, would we consider $\phi \wedge \psi$ to be well-defined? What if ψ is false but ϕ may be undefined? Is $\exists x\phi$ well-defined if there exists a (ground) term t such that $\phi(t/x)$ is well-defined and true? or should we insist that $\phi(x)$ is well-defined? If a subformula of the form $\phi \vee \neg\phi$ is encountered is it immediately evaluated to true, without decending into the recursive truth definition of ϕ?

The papers [5, 3] take another approach. They start with a 3-valued logic and call a formula ϕ well-defined if it can be proved that ϕ never evaluates to the third truth value *undefined*. But, this only shifts the problem. The notion of well-definedness now depend on the particular 3-valued logic in use and on the algorithm to compute the truth value. The paper [5] offers a choice between two versions of 3-valued logic. The paper [17] considerably improves the results from [5, 3]. We will say more on this below.

There is a strong feeling that the different definitions of well-definedness are in a sense approximations. But, approximations to what? Also all notions of well-definedness mentioned so far are not preserved by logical equivalence. Is there a sensible notion of well-definedness that is preserved by logical equivalence?

Our answer if given as Definition 12 below. It is a purely semantic definition and may roughly be paraphrased as:

> A formula ϕ is well-defined with respect to a theory Th if the truth value of ϕ in any model of Th does not depend on the values of functions f outside their fixed value formula fix_f.

To distinguish this concept from previous well-definedness notions we introduce a new terminology and call such ϕ *total*.

What we have discussed so far concerns formulas ϕ that are derived from a given background theory Th, i.e., $Th \vdash \phi$. But, what about the axioms of Th? Should they also be well-founded? In the theory of sequences described in Sections 2.7 and 4.2 e.g., the axiom

$$\forall seq\ s \forall int\ i, j(i > j \to seqLen(seqSub(s, i, j)) = 0)$$

is encountered. This formula is definitely not well-defined, since the fixed-value formula for $seqSub(s, i, j)$ requires $0 \leq i \leq j < seqLen(s)$. The theory for sequences for the Dafny system contains the axiom

$$\begin{array}{l} \forall < T > SeqT\ s \forall int\ i \forall T\ v \forall int\ len(\\ 0 <= len ==> Seq\#Length(Seq\#Build(s, i, v, len)) == len) \end{array}$$

The function $Seq\#Build(s, i, v, len)$ returns the subsequence of s from index 0 to index $len - 1$ that is in addition updated at index i by the value

v. This axiom is also not well-defined since the fixed-value formula for $Seq\#Build(s, i, v, len)$ is $(0 \leq \ len < Seq\#Length(s)) \wedge (0 \leq i < len)$.

Including the full domain restrictions would make the axioms considerably more bulky and reaoning with them, very likely, more involved. This is the motivation for this phenomenon that has been named *overspecification* since it fixes some function values outside the intended domain of the function. There is the feeling that overspecification will not hurt, since we are only interested in total (well-defined) consequences of the axioms. In the context of this report we believe that overspecification does not compromise the proofs in section 1.4. The claims (1.9) and (1.10) depend only on defined values, they are in some sense well-defined. We will introduce below, Definition 13, the notion of two theories T_1, T_2 coinciding on the fixed part of their common signature, that is to say the same total formulas can be derived from T_1 and T_2. This will allow us to compare a strictly defensive axiomatisation Ax_1 of some abstract data type with a more relaxed axiomatisation Ax_2 including overspecification. If Ax_1 and Ax_2 agree on the fixed part of their common signature then we are sure that the overspecification in Ax_2 does not cause problems. Lemma 13 below will provide a useful criterion to prove fixed-part-coincidence.

2.2 A Theory of Underspecification

Definition 6 (Partial Signature)
The symbols $\Sigma = \Sigma_p \cup \Sigma_t$ of a partial signature *are divided into partial, Σ_p and total symbols Σ_t.*

For every n-place symbol $f \in \Sigma_p$ a Σ_t-formula $fix_f(x_1, \ldots, x_n)$ with at most the free variables $x_1, \ldots, x_n$ is also part of the signature definition, called the fixed values formula *of f.*

In the following we will asume that Σ_p contains only function symbols. It is in fact hard to imagine natural examples of partial predicates. In any case in the Bellman-Ford case study partial predicates do not occur. If, nevertheless, there is a need for a partial predicate *pred* one could use its characteristic Boolean function f_{pred} ($pred(x_1, \ldots, x_n) \Leftrightarrow f_{pred}(x_1, \ldots, x_n) = \ true$) instead.

What we call here the fixed values formula $fix_f(x_1, \ldots, x_n)$ is in most other publications called a *domain restriction*. Since in the context of underspecification every function is total, there is strictly speaking no restriction of the domain. It seemed to us more appropriate to distinguish between values that are fixed and values that could also be otherwise. But, as with most terminological issues, this is a matter of taste.

Example 2

function	*fixed values formula*
$int\ w(node, node)$	$edge(x_1, x_2)$
$node :: seqGet(seq, int)$	$0 \leq x_2\ \&\ x_2 < seqLen(x_1)\ \&$ $node :: instance(any :: seqGet(x_1, x_2))$
$seq\ \ seqSub(seq, int, int)$	$0 \leq\ x_2\ \&\ x_2\ \leq\ x_3\ \&\ x_3\ <\ seqLen(x_1)$
$int\ /\ int$	$x_2 \neq 0$
$int\ pw(seq)$	$\forall i((0 \leq i\ \&\ i < seqLen(x_1) - 1) \rightarrow$ $edge(node :: seqGet(x_1, i),$ $(node :: seqGet(x_1, i+1)))$

We have adopted the requirement that fix_f be a Σ_t-formula to keep things simple. Anyhow the requirement is satisfied in all our examples. More liberal requirements where fix_f could contain partial function symbols will work, provided circular dependencies are avoided. In particular, fix_f should not contain f.

The adoption of a specific partial signature Σ already includes important modeling decisions. Consider as an example the bounded sum function $bsum\{int\ i\}(int\ from, int\ to, int\ t)$ as a formalisation of $\Sigma_{i=from}^{i<to} t(i)$. $bsum$ is a 3-place function symbol. The integer variable i plays a role similar to bound variables in quantification. One modeling alternative would be to declare $bsum\{int\ i\}(int\ from, int\ to, int\ t)$ to be undefined when $to \geq from$. The axiomatisation reviewed in Subsection 4.3 takes another approach. For inputs $from \geq to$ the value of $bsum$ is set to 0.

In the above Example 2 $seqSub(s, from, to)$ was declared to be undefined when $0 \leq from \leq to < seqLen(s)$ is false. A partial alternative would be to define $\forall Seq\ s \forall\ from,\ to(from > to \rightarrow seqSub(s, from, to) = seqEmtpy)$. This would still leave the cases with $from \leq to$ but $from < 0$ or $seqLen(s) \leq to$ undefined.

Definition 7 (Partial Structures)
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature.
A structure $\mathcal{M} = (M, I)$ is a partial Σ-structure if

1. *every n-place function symbol $f \in \Sigma_t$ is interpreted as a total function $I(f)$,*

2. *for every n-place function symbol $f \in \Sigma_p$*

$$fix_f^{\mathcal{M}} \subseteq dom(I(f)),$$

   with the abbreviation

$$fix_f^{\mathcal{M}} = \{(a_1, \ldots, a_n) \in M^n \mid (\mathcal{M} \upharpoonright \Sigma_t) \models fix_f[a_1, \ldots, a_n]\}.$$

3. *There are no restrictions on the interpretations $I(r)$ of predicate symbols $r \in \Sigma$.*

Furthermore, we have made use of the common concept and definition of the domain $dom(pf)$ for n-ary partial functions $pf : X^n \to Y$, $dom(pf) = \{(e_1, \ldots, e_n) \in X^n \mid pf(e_1, \ldots, e_n)$ is defined$\}$.

A partial structure $\mathcal{M} = (M, I)$ is called a total *structure if all functions $I(f)$ are total for all function symbols $f \in \Sigma$.*

In the previous definition we used the notation $\mathcal{N} \models \phi[a_1, \ldots, a_n]$ as a shorthand for $(\mathcal{N}, \beta) \models \phi$ with $\beta(x_i) = a_i$ for all $1 \leq i \leq n$. This abbreviation presupposes that that we know from the context which variable in ϕ will be assigned which value in the sequence $a_1, \ldots, a_n$. In all cases where we will use this shortcut this association will be obvious.

Definition 8 (Extensions of Partial Structures)
Let $\mathcal{M} = (M, I^{\mathcal{M}})$, $\mathcal{N} = (N, I^{\mathcal{N}})$ be two partial structuress for the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$.
$\mathcal{M}$ is an extension of $\mathcal{N}$ if

1. *$M = N$,*

2. *all symbols $f \in \Sigma_t$ have the same interpretation, $I^{\mathcal{M}}(f) = I^{\mathcal{N}}(f)$,*

3. *for all n-ary function symbols $f \in \Sigma_p$ and all n-tupels $(m_1, \ldots, m_n) \in M^n$ such that $I^{\mathcal{M}}(f)(m_1, \ldots, m_n)$ is defined, also $I^{\mathcal{N}}(f)(m_1, \ldots, m_n)$ is defined and*

$$I^{\mathcal{M}}(f)(m_1, \ldots, m_n) = I^{\mathcal{N}}(f)(m_1, \ldots, m_n)$$

Do not confuse the extension of a structure with the expansion of a structure from Definition 27.

We will later also need the dual notation of a reduction of a total structure to a partial structure. This seems a good place to introduce it.

Definition 9 (Reductions of Partial Structures)
Let $\mathcal{M} = (M, I^{\mathcal{M}})$ be a total structure of the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$. The partial reduction of *$\mathcal{M}$, denoted by $\mathcal{M}^p_\Sigma = (M, I^{\mathcal{M}}_p)$, if given by*

- *the same universe M as $\mathcal{M}$,*
- *the same interpretation of all symbols in Σ_t,*
- *the total function $I^{\mathcal{M}}(f)$ is replaced by the partial function $I^{\mathcal{M}}_p(f)$ that coincides with $I^{\mathcal{M}}(f)$ but is restricted to the domain*

  $$\{(a_1, \ldots, a_n) \in M^n \mid \mathcal{M} \models fix_f[a_1, \ldots, a_n]\}$$

  for all $f \in \Sigma_p$.

We note, that in general there may be a partial structure $\mathcal{N}^0$ and a total extension $\mathcal{M}$ of $\mathcal{N}^0$ such that $\mathcal{M}^p_\Sigma$ is different from $\mathcal{N}^0$. On the other hand

Lemma 3
Let $\mathcal{M}$ be a total Σ-structure for the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$ and $\mathcal{N}$ an arbitrary total extension of $\mathcal{M}^p_\Sigma$. Then

$$\mathcal{N}^p_\Sigma \quad = \mathcal{M}^p_\Sigma$$

Proof Easy computation.

■

Definition 10 (Partial Semantics)
Let $\mathcal{M} = (M, I)$ be a partial structure over the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$, ϕ a Σ-formula, and β a variables assignment.

We define ϕ to be true in $(\mathcal{M}, \beta)$, in symbols

$$(\mathcal{M}, \beta) \models_p \phi,$$

if $(\mathcal{N}, \beta) \models \phi$ for all total structures $\mathcal{N}$ that are extensions of $\mathcal{M}$.

We finish off the series of basic definitions by the obvious

Definition 11 (Partial Entailment)
Let Σ be a partial signature, Φ a set of Σ-formulas and ϕ a single Σ-formula.

1. *We say that ϕ is a partial logical consequence of Φ, in symbols*

   $$\Phi \vdash_p \phi,$$

   if for all partial Σ structures $\mathcal{M}$ and all variable assignments β

   $$\text{if } (\mathcal{M}, \beta) \models_p \Phi \text{ then } (\mathcal{M}, \beta) \models_p \phi$$

2. *We still write*

   $$\Phi \vdash \phi,$$

   for the usual logical consequence relation where Σ is considered as a normal signature with all symbols total.

The main advantages of this definition of logical consequence in the presence of partial structures are

1. we circumvented evaluation of terms and formulas in partial structures

2. we do not need a new calculus as Lemma 4 below shows.

Lemma 4
Let Σ be a partial structure, Φ a set of Σ-formulas and ϕ a single Σ-formula. Then

$$\Phi \vdash_p \phi \quad \Leftrightarrow \quad \Phi \vdash \phi$$

Proof We assume in this proof that Φ and ϕ contain no free variables. The same proof works for the general case by adding β everywhere.

Assume $\Phi \vdash \phi$ and let $\mathcal{M}$ be a partial Σ structure with $\mathcal{M} \models_p \Phi$. We need to show $\mathcal{M} \models_p \phi$. By definition of $\models_p$ we know for all total extensions $\mathcal{N}$ of $\mathcal{M}$ $\mathcal{N} \models \Phi$. Now, $\Phi \vdash \phi$ implies for all those $\mathcal{N}$ also $\mathcal{N} \models \phi$. This proves $\mathcal{M} \models_p \phi$.

For the converse we assume $\Phi \vdash_p \phi$ and look at a total structure $\mathcal{M}$ with $\mathcal{M} \models \Phi$ with the aim to show $\mathcal{M} \models \phi$. Since $\mathcal{M}$ is the only total structure that extends $\mathcal{M}$ we trivially have $\mathcal{M} \models_p \Phi$. By assumption we get $\mathcal{M} \models_p \phi$. Since $\mathcal{M}$ is already a total structure this is the same as $\mathcal{M} \models \phi$.

■

Theorems analogous to Lemma 4 have been proved in [1] for the propositional logic $S5$ (i.e., the modal logic where every world is accessible from every other world) and for a complete logic C with the same propositional modal syntax but semantics restricted to models where states are identified with mappings assigning truth values to propositional variables.

Lemma 4 can be vastly generalised.

Lemma 5 (Generalised Partial Entailment)
Let T be an arbitrary mapping such that $T(\mathcal{M})$ is a non-empty set of total structures for any partial structure $\mathcal{M}$. We only require that $T(\mathcal{M}) = \{\mathcal{M}\}$ if $\mathcal{M}$ is total.
Mimicking definitions 10 and 11 in this general context we define

$$\mathcal{M} \models_T \phi \quad \Leftrightarrow \quad \mathcal{N} \models \phi \text{ for all } \mathcal{N} \in T(\mathcal{M})$$

and

$$\begin{array}{lll} \Phi \vdash_T \phi & \Leftrightarrow & \text{for all models } \mathcal{M} \\ & & \text{if } \mathcal{M} \models_T \Phi \text{ then } \mathcal{M} \models_T \phi \end{array}$$

Then:

$$\Phi \vdash_T \phi \quad \Leftrightarrow \quad \Phi \vdash \phi$$

Proof Easy adaptation of the proof of Lemma 4.

∎

2.3 Total Formulas

Definition 12 (Total Formulas)
Let $\phi(x_1, \ldots, x_n)$ be a formula over a partial signature $\Sigma = \Sigma_p \cup \Sigma_t$, and Th a Σ-theory, i.e. a set of Σ-sentences.

A formula ϕ is called total *with respect to Th*

iff

for any (total) Σ-model $\mathcal{M}$ of Th (i.e. $\mathcal{M} \models Th$) and $(a_1, \ldots, a_n) \in M^n$

- *if $\mathcal{M} \models \phi[a_1, \ldots, a_n]$*
- *then*
  for any other (total) Σ-structure $\mathcal{N}$ satisfying $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$ (see Definition 9) we must also have $\mathcal{N} \models \phi[a_1, \ldots, a_n]$

We call a formula ϕ total *if it is* total *with respect to the empty theory.*

In the ensuing text we will tacitly use the following lemma.

Lemma 6
Let $\Sigma = \Sigma_p \cup \Sigma_t$ *be a partial signature,* Th *a* Σ*-theory.*
Every Σ_t*-formula is total with respect to* Th*.*

Proof Obvious.

∎

Typical examples of total formulas are $\exists x(f(x) = 0 \wedge fix_f(x))$, $\exists x(f(x) = g(x) \wedge fix_f(x) \wedge fix_g(x))$, or $\forall x(fix_f(x) \to f(x) > 0)$ if f, g are unary partial functions.

Lemma 7
Let Σ *be a partial signature, and* T *a* Σ*-theory*

1. *If* $T \vdash \phi$ *or* $T \vdash \neg\phi$ *then* ϕ *is total with respect to* T*.*
   In particular if ϕ *is a tautology or contradictory then* ϕ *is total.*
2. *If* ϕ,ψ *are total with respect to* T *so are*
   $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$, $\forall x\phi$ $\exists x\phi$.

Proof Item 1 is obviously true. Also item 2 is easy. Let us present just one case, closure under negation. We have to consider two full Σ models $\mathcal{M}$ and $\mathcal{N}$ of T such that $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$ and $\mathcal{M} \models \neg\phi$. If $\mathcal{N} \models \phi$ then we obtain from the assumption that ϕ is total and the observation that the definition of totality is symmterical in $\mathcal{M}$ and $\mathcal{N}$ that also $\mathcal{M} \models \phi$. A contradiction. Thus $\mathcal{N} \models \neg\phi$.

∎

By item 1 of Lemma 7 the formula $\frac{1}{0} = \frac{1}{0}$ is total. This is in the spirit of underspecification, undefined terms are not totally avoided, they are allowed whenever it does not make any difference how they are evaluated.

In general it is not so easy to find out if a formula is total. We will present an accessible criterion for totality below, Lemma 9. First however, we will continue with the outline of our approach at the top level.

The situation we are faced with can be described as follows: we are given two theories Th_1 and Th_2 in the same partial signature $\Sigma = \Sigma_p \cup \Sigma_t$ that deviate in the way they treat undefined values but agree on the fixed part of Σ. We expect that Th_1 and Th_2 agree on the total formulas. Here is a precise definition of what we mean by agreement on the fixed part of Σ,

Definition 13 (Agreement on Fixed Part)

Let Th_1 and Th_2 be theories in the same partial signature $\Sigma = \Sigma_p \cup \Sigma_t$. We say that Th_1 and Th_2 agree on the fixed part of Σ if for any total Σ-sentence ϕ

$$Th_1 \vdash \phi \quad \Leftrightarrow Th_2 \vdash \phi$$

Theorem 8
Let Th_1 and Th_2 be theories in the same partial signature $\Sigma = \Sigma_p \cup \Sigma_t$. Assume that

1. *for every Σ-structure $\mathcal{M}$ with $\mathcal{M} \models Th_1$ there is an extension $\mathcal{N}$ of the partial reduction $\mathcal{M}^p_\Sigma$ with $\mathcal{N} \models Th_2$*

and vice versa:

2. *for every Σ-structure $\mathcal{N}$ with $\mathcal{N} \models Th_2$ there is an extension $\mathcal{M}$ of the partial reduction $\mathcal{N}^p_\Sigma$ with $\mathcal{M} \models Th_1$*

Then Th_1 and Th_2 agree on the fixed part of Σ.

Proof Assume first that $Th_1 \vdash \phi$ and let $\mathcal{N}$ be an arbitrary model of Th_2, i.e., $\mathcal{N} \models Th_2$. Let $\mathcal{M}$ be a model of Th_1 that is an extension of $\mathcal{N}^p_\Sigma$ which exists by the assumption on Th_1 and Th_2. From $Th_1 \vdash \phi$ we obtain $\mathcal{M} \models \phi$. Since $\mathcal{N}^p_\Sigma = \mathcal{M}^p_\Sigma$ we get by the definition of a total formula also $\mathcal{N} \models \phi$. The reverse implication is proved symmetrically.

▪

In order for Theorem 8 to be useful we should have effective means to

1. determine wether a formula is total
2. see that two theories agree on the fixed part of their signature

We will address both issues in turn. First we return to the problem already aluded to above, is there an easy way to prove that a formula is total? As a prerequisite we need to deal with well-defined terms.

Definition 14 (Well-defined Terms)
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, t a Σ-term containing at most the variables $x_1, \ldots, x_n$. Let furthermore $f^i(t_{i,1}, \ldots, t_{i,k_i})$ for $0 \leq i \leq r$ be all subterms of t with leading function symbol $f^i \in \Sigma_p$. The formula $wf_t(x_1, \ldots, x_n)$ with at most the free variables $x_1, \ldots, x_n$, called the well-definedness predicate for t *is defined by:*

$$wf_t(x_1, \ldots, x_n) = \begin{cases} \bigwedge_{0 \leq i \leq r} fix_{f^i}(t_{i,1}, \ldots, t_{i,k_i}) & \text{if } r > 0 \\ true & \text{if } r = 0 \end{cases}$$

For the puposes of this definition we assume for t a variable or a total function symbol $fix_f \equiv true$.

The following definition gives a purely syntactic definition of a well-definedness predicate wd_ϕ for any Σ-formula ϕ.

Definition 15
For any formula $\phi(x_1, \ldots, x_n)$ over a partial signature $\Sigma = \Sigma_p \cup \Sigma_t$ we define a Σ_t-formula wd_ϕ as follows

1. $wd_\phi = \bigwedge_{1 \leq i \leq n} wf_{t_i}$ *if $\phi = p(t_1, \ldots, t_n)$ is an atomic formula*

2. $wd_{\neg\phi} = wd_\phi$

3. $wd_{\phi_1 \wedge \phi_2} = wd_{\phi_1} \wedge wd_{\phi_2}$, $wd_{\phi_1 \vee \phi_2} = wd_{\phi_1} \wedge wd_{\phi_2}$, $wd_{\phi_1 \to \phi_2} = wd_{\phi_1} \wedge wd_{\phi_2}$

4. $wd_{\forall x \phi} = \begin{cases} \forall x(\phi_0 \to wd_{\phi_1}) & \text{if } \phi \equiv \phi_0 \to \phi_1 \text{ with } \phi_0 \text{ a } \Sigma_t\text{-formula} \\ \forall x wd_\phi & \text{otherwise} \end{cases}$

5. $wd_{\exists x \phi} = \begin{cases} \forall x(\phi_0 \to wd_{\phi_1}) & \text{if } \phi \equiv \phi_0 \wedge \phi_1 \text{ with } \phi_0 \text{ a } \Sigma_t\text{-formula} \\ \forall x wd_\phi & \text{otherwise} \end{cases}$

The formula wd_ϕ strongly depends on the accidental syntactic structure of ϕ. The wd formula for $\forall x(\phi_1 \to \phi_2)$ is different from the wd formula for $\forall x(\neg\phi_1 \vee \phi_2)$. We can thus only expect that Lemma 9 and Corollary 10 below are sufficient criteria and by no means necessary. This leaves considerable room for variations of Definition 15. It might e.g., be a good idea to define weaker well-definedness predicates that are easier to check and do not sacrifice much on precision. This is done in [7, 17]. This paper also contains a carefully researched list of previous research on the use of partial functions in formal specification and verification, e.g, [3, 5].

In [13, 14] a variation of the *wd* formula, called the *Exact* predicate, is given in the context of behavioral interface specification languages (at that time the prototypical example of a BISL was the behavioral part of LSL, the Larch specification language a precursor of present-day JML). The *Exact* predicate differs in the inductive definition steps for the quantifiers, where it does not take into account the part denoted by ϕ_0 in Definition 15. Also, what we called the *fixed value formula* is not supplied as part of the signature but more flexible through specification clauses.

Lemma 9 (Totality Criterion)
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, and wd_ϕ be as in Definition 15. Then for any (total) Σ-structure $\mathcal{M}$ and $(a_1, \ldots, a_n) \in M^n$ the following is true:

- *if $\mathcal{M} \models wd_\phi[a_1, \ldots, a_n]$*
- *then*
  for any other (total) Σ-structure $\mathcal{N}$ satisfying $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$ (see Definition 9) we must have

$$\mathcal{M} \models \phi[a_1, \ldots, a_n] \Leftrightarrow \mathcal{N} \models \phi[a_1, \ldots, a_n]$$

The claim of this lemma sounds rather plausible. Nevertheless, let us step through a detailed proof.

Proof By structural induction on the formula ϕ.
(**terms**) As a preparatory step we show for any Σ-term t:
Let $\mathcal{M}$ be an arbitrary (total) Σ-structure, $(a_1, \ldots, a_n) \in M^n$ with

$$\mathcal{M} \models wf_t[a_1, \ldots, a_n].$$

Let $\mathcal{N}$ be another (total) Σ-structure satisfying $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$. Then we claim:

$$t^{\mathcal{M}}(a_1, \ldots, a_n) = t^{\mathcal{N}}(a_1, \ldots, a_n)$$

This is proved by induction on the complexity of the, possibly nested, term t. Let us just consider the case that $t \equiv f(x_1, \ldots, x_m)$ and leave the rest of the argument to the reader.

The interptretation of a total function f is unchanged in all four structures $\mathcal{M}$, $\mathcal{M}^p_\Sigma$, $\mathcal{N}^p_\Sigma$, and $\mathcal{N}$.

If $f \in \Sigma_p$ the assumption guarantees that the tupel $(a_1, \ldots, a_n)$ is in the fixed domain of f (remember that in this simple case $wf_t \equiv fix_f$) thus the value of $f(a_1, \ldots, a_n)$

- is the same in $\mathcal{M}$ and $\mathcal{M}^p_\Sigma$,
- is the same in $\mathcal{M}^p_\Sigma$ and $\mathcal{N}^p_\Sigma$
  since $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$
- is the same in $\mathcal{N}^p_\Sigma$ and $\mathcal{N}$

This guarantees:

$$f^{\mathcal{M}}(a_1, \ldots, a_n) = f^{\mathcal{N}}(a_1, \ldots, a_n)$$

(**atomic formulas**) Let $\phi = p(t_1, \ldots, t_m)$ for a predicate symbol p. Assume $\mathcal{M} \models wd_\phi[a_1, \ldots, a_m]$ and $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$. By definition $wd_\phi = \bigwedge_{1 \leq i \leq m} wf_{t_i}$. As shown in the first paragraph of this proof $t_i^{\mathcal{M}}[\bar{a}] = t_i^{\mathcal{N}}[\bar{a}]$. Thus

$$\begin{aligned}
\mathcal{M} \models p(t_1, \ldots, t_m)[\bar{a}] &\Leftrightarrow (t_1^{\mathcal{M}}[\bar{a}], \ldots, t_m^{\mathcal{M}}[\bar{a}]) \in p^{\mathcal{M}} && \text{semantics def.} \\
&\Leftrightarrow (t_1^{\mathcal{N}}[\bar{a}], \ldots, t_m^{\mathcal{N}}[\bar{a}]) \in p^{\mathcal{M}} && \text{see above} \\
&\Leftrightarrow (t_1^{\mathcal{N}}[\bar{a}], \ldots, t_m^{\mathcal{N}}[\bar{a}]) \in p^{\mathcal{N}} && \text{total predicates} \\
&\Leftrightarrow \mathcal{N} \models p(t_1, \ldots, t_m)[\bar{a}] && \text{semantics def.}
\end{aligned}$$

(**propositional connectives**) These cases are all straigth forward. We only present that case of a disjunction in detail. We suppress the assigment $[\bar{a}]$ of free variables.
Assumption $\mathcal{M} \models wd_{\phi_1 \vee \phi_2}$ with $wd_{\phi_1 \vee \phi_2} = wd_{\phi_1} \wedge wd_{\phi_2}$.

$$\begin{aligned}
\mathcal{M} \models \phi_1 \vee \phi_2 &\Leftrightarrow \mathcal{M} \models \phi_i \text{ for some } i && \text{semantics definition} \\
&\Leftrightarrow \mathcal{N} \models \phi_i && \text{Ind.Hyp. using } \mathcal{M} \models wd_{\phi_i} \\
&\Leftrightarrow \mathcal{N} \models \phi_1 \vee \phi_2 && \text{semantics definition}
\end{aligned}$$

(**universal quantification**) Let $\phi = \forall x_0 \phi_0(x_0, \ldots, x_m)$. We first consider the case $wd_\phi = \forall x_0 wd_{\phi_0}$. As an assumption we have $\mathcal{M} \models \forall x_0 wd_{\phi_0}[\bar{a}]$ for $\bar{a} = a_1, \ldots, a_m$.

$$\begin{aligned}
\mathcal{M} \models \forall x_0 \phi_0[\bar{a}] &\Leftrightarrow \mathcal{M} \models \phi_0[a, \bar{a}] \quad \text{for all } a \in M && \text{semantics definition} \\
&\Leftrightarrow \mathcal{N} \models \phi_0[a, \bar{a}] \quad \text{for all } a \in M && \text{Ind.Hyp. and} \\
& && \mathcal{M} \models wd_{\phi_0}[a, \bar{a}] \\
&\Leftrightarrow \mathcal{N} \models \forall x_0 \phi_0[\bar{a}] && \text{semantics definition}
\end{aligned}$$

Now consider the case $\phi = \forall x_0(\phi_0 \to \phi_1)(x_0, \bar{x})$ with ϕ_0 a Σ_t-formula. Remember, we have $wd_\phi = \forall x_0(\phi_0 \to wd_{\phi_1})$.

$$\mathcal{M} \models \phi[\bar{a}] \quad \Leftrightarrow \quad \mathcal{M} \models \phi_0 \to \phi_1[a, \bar{a}] \quad \text{for all } a \in M \qquad \text{semantics def.}$$

If $\mathcal{M} \models \neg\phi_0[a, \bar{a}]$ then $\mathcal{N} \models \neg\phi_0[a, \bar{a}]$ since ϕ_0 is a Σ_t-formula and we immediately get

$$\Leftrightarrow \quad \mathcal{N} \models \phi_0 \to \phi_1[a, \bar{a}] \quad a \in M$$

In case $\mathcal{M} \models \phi_0[a, \bar{a}]$ we also have $\mathcal{M} \models \phi_1[a, \bar{a}]$ and $\mathcal{M} \models wd_{\phi_1}[a, \bar{a}]$. Now, we can appeal to the induction hypothesis to obtain $\mathcal{N} \models \phi_1[a, \bar{a}]$. In total we have now shown

$$\mathcal{M} \models \phi[\bar{a}] \quad \Leftrightarrow \quad \mathcal{N} \models \phi[\bar{a}]$$

(**existential quantification**) Let $\phi = \exists x_0 \phi_0(x_0, \ldots, x_m)$. We first consider the case $wd_\phi = \forall x_0 wd_{\phi_0}$.

$$\begin{array}{rcll} \mathcal{M} \models \exists x_0 \phi_0[\bar{a}] & \Leftrightarrow & \mathcal{M} \models \phi_0[a, \bar{a}] \quad \text{for some } a \in M & \text{semantics def.} \\ & \Leftrightarrow & \mathcal{N} \models \phi_0[a, \bar{a}] \quad \text{for some } a \in M & \text{Ind.Hyp and} \\ & & & \mathcal{M} \models wd_{\phi_0}[a, \bar{a}] \\ & \Leftrightarrow & \mathcal{N} \models \exists x_0 \phi_0[\bar{a}] & \end{array}$$

Notice, that the induction hypothesis is applicable because initially we have $\mathcal{M} \models wd_\phi[\bar{a}]$. Since $wd_\phi = \forall x_0 wd_{\phi_0}$, we obtain $\mathcal{M} \models wd_{\phi_0}[a, \bar{a}]$ for the existing a. Had we defined $wd_\phi^{wrong} = \exists x_0 wd_{\phi_0}$, the argument would not have worked.

Finally, let us take up the case $\phi = \exists x_0(\phi_0 \ \& \ \phi_1(x_0, \ldots, x_m))$ with ϕ_0 a Σ_t-formula and case $wd_\phi = \forall x_0(\phi_0 \to wd_{\phi_1})$.

$$\mathcal{M} \models \exists x_0(\phi_0 \ \& \ \phi_1)[\bar{a}] \quad \Leftrightarrow \quad \mathcal{M} \models (\phi_0 \ \& \ \phi_1)[a, \bar{a}] \quad \text{for some } a \in M$$
semantics definition

From $\mathcal{M} \models \phi_0[a, \bar{a}]$
We get $\mathcal{N} \models \phi_0[a, \bar{a}]$ since ϕ_0 is a Σ_t-formula
From $\mathcal{M} \models \phi_0[a, \bar{a}]$
We also get $\mathcal{M} \models wd_{\phi_1}[a, \bar{a}]$ since $wd_\phi = \forall x_0(\phi_0 \to wd_{\phi_1})$
Thus by Ind.Hyp $\mathcal{N} \models \phi_1[a, \bar{a}]$

$$\begin{array}{cl} \Leftrightarrow & \mathcal{N} \models \phi_0 \ \& \ \phi_1[a, \bar{a}] \\ \Leftrightarrow & \mathcal{N} \models \exists x_0(\phi_0 \ \& \ \phi_1)[\bar{a}] \end{array}$$

∎

Corollary 10
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, Th a Σ-theory and wd_ϕ be as in Definition 15 with the free variables $\bar{x} = x_1, \ldots, x_n$.
If $Th \vdash \forall \bar{x} wd_\phi$ then ϕ is total with respect to Th.

Proof Follows easily from Lemma 9.

∎

The next lemma offers a (very) partial converse of Lemma 9. It also gives support to our claim that total formulas are the right abstraction of well-definedness.

Lemma 11
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, f an n-place function symbol in Σ_t, $t_1, \ldots, t_n, t_{n+1}$ terms that do not contain the symbol f, $(x_1, \ldots, x_k) = \bar{x}$ all variables occuring in t_1 to t_{n+1}. If

$$\forall \bar{x}(f(t_1, \ldots, t_n) = t_{n+1}) \textit{ is a total formula}$$

then

$$\forall \bar{x}(fix_f(t_1, \ldots, t_n)) \textit{ is a tautology}$$

Proof If $\forall \bar{x}(fix_f(t_1, \ldots, t_n))$ is not a tautology there is a Σ-structure $\mathcal{M} = (M, I)$ and a variable assignment β with $(\mathcal{M}, \beta) \models \neg fix_f(t_1, \ldots, t_n)$. We can thus find an extension $\mathcal{M}' = (M, I')$ of $\mathcal{M}^p_\Sigma$ that differs from $\mathcal{M}$ only in the interpretation of the symbol f

$$I'(f)(a_1, \ldots, a_n) = \begin{cases} b & \text{if } a_i = t_i^{(\mathcal{M},\beta)} \text{ for all } i \\ I(f)(a_1, \ldots, a_n) & \text{otherwise} \end{cases}$$

In case $(\mathcal{M}, \beta) \models f(t_1, \ldots, t_n) = t_{n+1}$ is true chose with $b \neq t_{n+1}^{(\mathcal{M},\beta)}$. This entail $(\mathcal{M}', \beta) \models f(t_1, \ldots, t_n) \neq t_{n+1}$ contradicting assumed the totality of $\forall \bar{x}(f(t_1, \ldots, t_n) = t_{n+1}$. Notice, that $t_i^{(\mathcal{M},\beta)} = t_i^{(\mathcal{M}',\beta)}$ for all $1 \leq i \leq n+1$ since f does not occur in any of these terms.

In case $(\mathcal{M}, \beta) \models f(t_1, \ldots, t_n) \neq t_{n+1}$ chose $b = t_{n+1}^{(\mathcal{M},\beta)}$. Thus $(\mathcal{M}', \beta) \models f(t_1, \ldots, t_n) = t_{n+1}$. This again contradicts the totallity of $\forall \bar{x}(f(t_1, \ldots, t_n) = t_{n+1})$.

∎

The relativization of the previous Lemma 11 to formulas that are total with respect to a particular theory T is not true in general. But, we have the following:

Lemma 12
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, T a Σ theory axiomatized by a set of total formulas, f an n-place function symbol in Σ_t, $t_1, \dots, t_n, t_{n+1}$ terms that do not contain the symbol f, $(x_1, \dots, x_k) = \bar{x}$ all variables occuring in t_1 to t_{n+1}. If

$$\forall \bar{x}(f(t_1, \dots, t_n) = t_{n+1}) \text{ is a total formula with respect to } T$$

then

$$T \vdash \forall \bar{x}(fix_f(t_1, \dots, t_n))$$

Proof Easy adaption of the proof of Lemma 11.

∎

The following lemma gives a criterion for two theories to agree on the fixed part of their common vocabulary (see Definition 13). It is a special case of Theorem 8 and is taylored towards a frequently occuring situation in the axiomatisation of abstract data types with partial functions.

Lemma 13
Let Th_1, Th_2 be two theories in a common partial signature $\Sigma = \Sigma_p \cup \Sigma_t$ and $\mathcal{P}$ a partial Σ-structure such that

1. *The partial reduction (Def. 9) of any Σ-model $\mathcal{M}$ of Th_1 or Th_2 equals $\mathcal{P}$, i.e., $\mathcal{M}^p_\Sigma = \mathcal{P}$,*
2. *There is a (total) extention $\mathcal{M}_2$ of $\mathcal{P}$ with $\mathcal{M}_2 \models Th_2$*
3. *For every (total) extention $\mathcal{M}_1$ of $\mathcal{P}$ we have $\mathcal{M}_1 \models Th_1$*

then
Th_1 and Th_2 agree on the fixed part of Σ.

Proof According to Theorem 8 we need to show

1. for every Σ-structure $\mathcal{M}$ with $\mathcal{M} \models Th_1$ there is an extension $\mathcal{N}$ of the partial reduction $\mathcal{M}^p_\Sigma$ with $\mathcal{N} \models Th_2$
2. for every Σ-structure $\mathcal{N}$ with $\mathcal{N} \models Th_2$ there is an extension $\mathcal{M}$ of the partial reduction $\mathcal{N}^p_\Sigma$ with $\mathcal{M} \models Th_1$

For the proof of 1. we note that $\mathcal{M}^p_\Sigma = \mathcal{F}$ by assumption 1 and the required extensions $\mathcal{N}$ exists by assumption 2. For the proof of 2. we note again that $\mathcal{N}^p_\Sigma = \mathcal{F}$ by assumption 3 every total extention of $\mathcal{N}$ can serve as a model for Th_1.

■

2.4 The Transformation $\mathcal{Y}$

Figure 2.1 gives the definition of the well-definedness predicate $\mathcal{Y}(\phi)$ from the paper [7].

$$\begin{array}{lcllcl}
\mathcal{T}(p(t_1,\ldots,t_n)) = p(t_1,\ldots,t_n) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i} & & & & & \\
 & & & \mathcal{F}(p(t_1,\ldots,t_n)) = \neg p(t_1,\ldots,t_n) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i} & & \\
\mathcal{T}(\neg\phi) & = & \mathcal{F}(\phi) & \mathcal{F}(\neg\phi) & = & \mathcal{T}(\phi) \\
\mathcal{T}(\phi_1 \wedge \phi_2) & = & \mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2) & \mathcal{F}(\phi_1 \wedge \phi_2) & = & \mathcal{F}(\phi_1) \vee \mathcal{F}(\phi_2) \\
\mathcal{T}(\phi_1 \vee \phi_2) & = & \mathcal{T}(\phi_1) \vee \mathcal{T}(\phi_2) & \mathcal{F}(\phi_1 \vee \phi_2) & = & \mathcal{F}(\phi_1) \wedge \mathcal{F}(\phi_2) \\
\mathcal{T}(\forall x\phi) & = & \forall x\mathcal{T}(\phi) & \mathcal{F}(\forall x\phi) & = & \exists x\mathcal{F}(\phi) \\
\mathcal{T}(\exists x\phi) & = & \exists x\mathcal{T}(\phi) & \mathcal{F}(\exists x\phi) & = & \forall x\mathcal{F}(\phi)
\end{array}$$

$$\mathcal{Y}(\phi) = \mathcal{T}(\phi) \vee \mathcal{F}(\phi)$$

Figure 2.1: The well-definedness predicate $\mathcal{Y}$ from [7]

The following lemma is contained in [7], in fact it lies at the heart of the introduction of the operators $\mathcal{T}$ and $\mathcal{F}$. We repeat it here for the reader's convenience.

Lemma 14
Let $\mathcal{Y}(\phi)$ as defined in Figure 2.1. Then

$$\mathcal{T}(\phi) \to \phi \quad \textit{and} \quad \mathcal{F}(\phi) \to \neg\phi$$

are tautologies.
Free variables in $\mathcal{T}(\phi) \to \phi$ and $\mathcal{F}(\phi) \to \neg\phi$ are implicitly universally quantified.

Proof The proof proceeds by induction on the structural complexity of ϕ. We present the induction step from ϕ to $\exists x\phi$. The induction hypotheses are

$$\mathcal{T}(\phi) \to \phi \quad \text{and} \quad \mathcal{F}(\phi) \to \neg\phi$$

By definition $\mathcal{T}(\exists x\phi) = \exists x\mathcal{T}(\phi)$. From the hypothesis we get $\exists x\mathcal{T}(\phi) \to \exists x\phi$ and this immediately $\mathcal{T}(\exists x\phi) \to \exists x\phi$.
Againy by definition of $\mathcal{F}$ we have $\mathcal{F}(\exists x\phi) = \forall x\mathcal{F}(\phi)$ while the hypothesis entails $\forall x\mathcal{F}(\phi) \to \forall x\neg\phi$. Together we get immediately $\mathcal{F}(\exists x\phi) \to \neg\exists x\phi$. The remaining cases are left to the reader or we refer to [7].

■

Lemma 15

1. *For any ϕ the formulas $\mathcal{T}(\phi), \mathcal{F}(\phi), \mathcal{Y}(\phi)$ are total.*
2. *We consider two full Σ models $\mathcal{M}$ and $\mathcal{N}$ of T such that $\mathcal{M}^p_\Sigma = \mathcal{N}^p_\Sigma$. If $\mathcal{M} \models \phi \wedge \mathcal{Y}(\phi)$ then $\mathcal{N} \models \phi$.*
3. *If $T \vdash \mathcal{Y}(\phi)$ then ϕ is total with respect to T.*

Proof
Claim (1) is proved by simultaneous induction on the complexity of ϕ. If ϕ is an atomic formula $p(t_1, \ldots, t_2)$ then $\mathcal{T}(p(t_1, \ldots, t_n)) = p(t_1, \ldots, t_n) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i}$, $\mathcal{F}(p(t_1, \ldots, t_n)) = \neg p(t_1, \ldots, t_n) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i}$ are obviously total formulas. In the following we will tacitly use item 2 of Lemma 7. The first appeal to this lemma yields totality of $\mathcal{Y}(\phi)$ for atomic ϕ. If $\phi = \phi_1 \wedge \phi_2$ we need to show that $\mathcal{T}(\phi) = \mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2)$ is total with respect to T. By induction hypothesis $\mathcal{T}(\phi_1)$ and $\mathcal{T}(\phi_2)$ are total thus also $\mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2)$. Along the same lines totality of $\mathcal{F}(\phi_1 \wedge \phi_2)$ and $\mathcal{Y}(\phi_1 \wedge \phi_2)$ is shown. Also the induction step for $\phi = \phi_1 \vee \phi_2$ does not pose any problem. In the step case $\phi = \neg\phi_0$ we have by simultaneous induction the totality if $\mathcal{T}(\phi_0)$,$\mathcal{F}(\phi_0)$, and $\mathcal{Y}(\phi_0)$ at our disposal. Since $\mathcal{T}(\neg\phi_0) = \mathcal{F}(\phi_0)$ and vice versa $\mathcal{F}(\neg\phi_0) = \mathcal{T}(\phi_0)$ the argument for this case is again completed. Finally, appealing to item 2 of Lemma 7 also the induction step for both quantifier cases are seen to be true.
Claim (2): The proof again proceeds by induction on the complexity of ϕ.
atomic case If ϕ is the atomic formula $p(t_1, \ldots, t_2)$ then

$$\begin{array}{rcl}
\mathcal{Y}(p(t_1, \ldots, t_n)) & = & \mathcal{T}(p(t_1, \ldots, t_n) \vee \mathcal{F}(p(t_1, \ldots, t_n)) \\
& = & (p(\bar{t}) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i}) \vee (\neg p(\bar{t}) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i}) \\
& \Leftrightarrow & (p(\bar{t}) \vee \neg p(\bar{t})) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i} \\
& \Leftrightarrow & \bigwedge_{i=1}^{i=n} wf_{t_i}
\end{array}$$

So if $\mathcal{M} \models p(\bar{t}) \wedge \mathcal{Y}(p(\bar{t}))$ then also $\mathcal{M} \models p(\bar{t})$.

conjunction We start from $\mathcal{M} \models (\phi_1 \wedge \phi_2) \wedge \mathcal{Y}(\phi_1 \wedge \phi_2)$. By Lemma 14 we must have $\mathcal{M} \models \mathcal{T}(\phi_1 \wedge \phi_2)$ since $\mathcal{M} \models \mathcal{F}(\phi_1 \wedge \phi_2)$ would lead to the contradiction $\mathcal{M} \models (\phi_1 \wedge \phi_2) \wedge \neg(\phi_1 \wedge \phi_2)$. This entails $\mathcal{M} \models \phi_i \wedge \mathcal{T}(\phi_i)$ and thus $\mathcal{M} \models \phi_i \wedge \mathcal{Y}(\phi_i)$ for $i = 1, 2$. Induction hypothesis yields $\mathcal{N} \models \phi_i \wedge \phi_2$, as desired.

disjunction Absolutely analogous to the conjunction case.

negation We start from $\mathcal{M} \models \neg\phi \wedge \mathcal{Y}(\neg\phi)$. By Lemma 14 we get $\mathcal{M} \models \mathcal{T}(\neg\phi)$ which yields by definition of $\mathcal{T}$ that $\mathcal{M} \models \neg\phi \wedge \mathcal{F}(\phi)$. By item (1) of this lemma we get $\mathcal{N} \models \mathcal{F}(\phi)$ and thus also $\mathcal{N} \models \mathcal{Y}(\phi)$. If $\mathcal{N} \models \phi$ were true we would obtain from the induction hypothesis and the symmetry with respect to $\mathcal{M}$ and $\mathcal{N}$ that $\mathcal{M} \models \phi$. A contradiction. Thus $\mathcal{N} \models \neg\phi$.

quantification We start from $\mathcal{M} \models \forall x\phi \wedge \mathcal{Y}(\forall x\phi)$. Following the pattern from previous cases we argue that we must have $\mathcal{M} \models \mathcal{T}(\forall x\phi)$. Since $\mathcal{M} \models \mathcal{F}(\forall x\phi)$ would produce the contradiction $\mathcal{M} \models (\forall x\phi) \wedge (\neg\forall x\phi)$. By definition of $\mathcal{T}$ we get $\mathcal{M} \models \forall x\mathcal{T}(\phi)$. For any element a in the common universe of $\mathcal{M}$ and $\mathcal{N}$ we get $\mathcal{M} \models \phi[a]$ and $\mathcal{M} \models \mathcal{T}(\phi)[a]$. Induction hypothesis now yields $\mathcal{N} \models \phi[a]$. Since this works for arbitrary a we have $\mathcal{N} \models \forall x\phi$. The case of the existential quantifier is absolutely analogous.

Claim (3): Trivial consequence of claim (2).

∎

Definition 16
By $PSubst(\phi)$ we denote the formula that arises from ϕ by substituting

- *every positive occurence of an atomic subformula $p(t_1, \ldots, t_n)$ in ϕ by $\bigwedge_{i=1}^{i=n} wf_{t_i} \wedge p(t_1, \ldots, t_n)$*
- *every negative occurence of an atomic subformula $p(t_1, \ldots, t_n)$ in ϕ by $\bigwedge_{i=1}^{i=n} wf_{t_i} \to p(t_1, \ldots, t_n)$.*

By $PSubst^{-}(\phi)$ we denote the following inverted substitution:

- *every negative occurence of an atomic subformula $p(t_1, \ldots, t_n)$ in ϕ is substituted by $\bigwedge_{i=1}^{i=n} wf_{t_i} \wedge p(t_1, \ldots, t_n)$*
- *every positive occurence of an atomic subformula $p(t_1, \ldots, t_n)$ in ϕ is substituted by $\bigwedge_{i=1}^{i=n} wf_{t_i} \to p(t_1, \ldots, t_n)$.*

An occurence of a subformula is called positive *if it is on the scope of an even number of negation symbols and* negative *if it is on the scope of an* *odd number of negation symbols. We trust that this explanation is precise enough. For a formal definition we would need to introduce a notation system to denote occurences within a formula.*

Lemma 16
The following equivalences are tautologies

$$
\begin{array}{rlcl}
1 & PSubst(\phi_1 \wedge \phi_2) & \Leftrightarrow & PSubst(\phi_1) \wedge\ PSubst(\phi_2) \\
 & PSubst^-(\phi_1 \wedge \phi_2) & \Leftrightarrow & PSubst^-(\phi_1) \wedge\ PSubst^-(\phi_2) \\
2 & PSubst(\phi_1 \vee \phi_2) & \Leftrightarrow & PSubst(\phi_1) \vee\ PSubst(\phi_2) \\
 & PSubst^-(\phi_1 \vee \phi_2) & \Leftrightarrow & PSubst^-(\phi_1) \vee\ PSubst^-(\phi_2) \\
3 & PSubst(\forall x\phi) & \Leftrightarrow & \forall x\ PSubst(\phi) \\
 & PSubst^-(\forall x\phi) & \Leftrightarrow & \forall x\ PSubst^-(\phi) \\
4 & PSubst(\exists x\phi) & \Leftrightarrow & \exists x\ PSubst(\phi) \\
 & PSubst^-(\exists x\phi) & \Leftrightarrow & \exists x\ PSubst^-(\phi) \\
5 & PSubs(\neg\phi) & \Leftrightarrow & \neg PSubs^-(\phi) \\
 & PSubs^-(\neg\phi) & \Leftrightarrow & \neg PSubs(\phi) \\
6 & PSubst(\neg(\phi_1 \vee \phi_2)) & \Leftrightarrow & PSubst(\neg\phi_1) \wedge\ PSubst(\neg\phi_2) \\
7 & PSubst(\neg(\phi_1 \wedge \phi_2)) & \Leftrightarrow & PSubst(\neg\phi_1) \vee\ PSubst(\neg\phi_2) \\
8 & PSubst(\neg\forall x\phi) & \Leftrightarrow & PSubst(\exists x\neg\phi) \\
9 & PSubst(\neg\exists x\phi) & \Leftrightarrow & PSubst(\forall x\neg\phi) \\
10 & PSubst(\neg\neg\phi) & \Leftrightarrow & PSubst(\phi)
\end{array}
$$

Proof A really formal proof would need a notation system for occurences of subformulas. But, we trust that the truth of the equivalences 1 to 5 will be accepted by the reader without this formal effort. The remaining claims 6 to 10 follow from 1 to 5 easily. Here is a proof of 7

$$
\begin{array}{rcl}
PSubst(\neg(\phi_1 \wedge \phi_2)) & \Leftrightarrow & \neg PSubst^-(\phi_1 \wedge \phi_2) \\
 & \Leftrightarrow & \neg(PSubst^-(\phi_1) \wedge\ PSubst^-(\phi_2)) \\
 & \Leftrightarrow & \neg PSubst^-(\phi_1) \vee \neg PSubst^-(\phi_2) \\
 & \Leftrightarrow & PSubst(\neg\phi_1) \vee PSubst(\neg\phi_2)
\end{array}
$$

∎

It is tempting to conjecture that for a tautology ϕ also $PSubst(\phi)$ is a tautology. But, $p(t)\vee\neg p(t)$ is always a tautology while $PSubst(p(t)\vee\neg p(t)) = (p(t) \wedge wf_t) \vee (\neg p(t)) \wedge wf_t) \Leftrightarrow wf_t$ is in general not.

Lemma 17
Let ϕ be a Σ-formula then

1. $\mathcal{T}(\phi) \Leftrightarrow PSubst(\phi)$

2. $\mathcal{F}(\phi) \Leftrightarrow PSubst(\neg\phi)$

Proof The proof proceeds by simultaneous induction on the complexity of ϕ.

atomic case

$$\begin{array}{rcl} \mathcal{T}(p(t_1,\ldots,t_n)) & = & \bigwedge_{i=1}^{i=n} wf_{t_i} \wedge p(t_1,\ldots,t_n) \\ & = & PSubst(p(t_1,\ldots,t_n)) \end{array}$$

$$\begin{array}{rcl} \mathcal{F}(p(t_1,\ldots,t_n)) & = & \bigwedge_{i=1}^{i=n} wf_{t_i} \wedge \neg p(t_1,\ldots,t_n) \\ & = & \neg(\bigwedge_{i=1}^{i=n} wf_{t_i} \rightarrow p(t_1,\ldots,t_n) \\ & = & PSubst(\neg p(t_1,\ldots,t_n)) \end{array}$$

conjunction

$$\begin{array}{rcll} \mathcal{T}(\phi_1 \wedge \phi_2) & = & \mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2) & \text{Def. } \mathcal{T} \\ & = & PSubst(\phi_1) \wedge PSubst(\phi_2) & \text{Ind.Hyp.} \\ & = & PSubst(\phi_1 \wedge \phi_2) & \text{property of } Psubst \end{array}$$

$$\begin{array}{rcll} \mathcal{F}(\phi_1 \wedge \phi_2) & = & \mathcal{F}(\phi_1) \vee \mathcal{F}(\phi_2) & \text{Def. } \mathcal{F} \\ & = & PSubst(\neg\phi_1) \vee PSubst(\neg\phi_2) & \text{Ind.Hyp.} \\ & = & PSubst(\neg\phi_1 \vee \neg\phi_2) & \text{property of } Psubst \\ & = & PSubst(\neg(\phi_1 \wedge \phi_2)) & \text{property of } Psubst \end{array}$$

disjunction Analogous to the conjunction case.

negation

$$\begin{array}{rcll} \mathcal{T}(\neg\phi) & = & \mathcal{F}(\phi_1) & \text{Def. } \mathcal{T} \\ & = & PSubst(\neg\phi) & \text{Ind.Hyp.} \\ \mathcal{F}(\neg\phi) & = & \mathcal{T}(\phi_1) & \text{Def. } \mathcal{F} \\ & = & PSubst(\phi) & \text{Ind.Hyp.} \\ & = & PSubst(\neg\neg\phi) & \text{property of } Psubst \end{array}$$

quantification

$$\begin{array}{rcll} \mathcal{T}(\forall x\phi) & = & \forall x \mathcal{T}(\phi) & \text{Def. } \mathcal{T} \\ & = & \forall x \; PSubst(\phi) & \text{Ind.Hyp.} \\ & = & PSubst(\forall x\phi) & \text{property of } Psubst \\ \mathcal{F}(\forall x\phi) & = & \exists x \mathcal{F}(\phi) & \text{Def.} \\ & = & \exists x \; PSubst(\neg\phi) & \text{Ind.Hyp.} \\ & = & PSubst(\neg\forall x\phi) & \text{property of } Psubst \end{array}$$

∎

For the convenience of the reader we repeat here the connection of the operator $\mathcal{Y}$ with three-valued logic. We assume a basic familiarity with three-valued logics.

Definition 17 (Three-valued Structure)
Let $\mathcal{M} = (M, I)$ be a (full) Σ-structure for a partial signature Σ. A 3-valued structure $\mathcal{M}^3 = (M^3, I^3)$ is defined by:

$$
\begin{array}{lcl}
M^3 & = & M \cup \{\bot\}, \bot \notin M \\
I^3(f)(a_1, .., a_n) & = & \begin{cases} I(f)(a_1, .., a_n) & \textit{if } (a_1, .., a_n) \in M^n \textit{ and} \\ & f \in \Sigma_t \textit{ or } \mathcal{M} \models fix_f[a_1, .., a_n] \\ \bot & \textit{otherwise} \end{cases} \\
I^3(p)(a_1, .., a_n) & = & \begin{cases} 1 & \textit{if } (a_1, .., a_n) \in M^n \textit{ and } \mathcal{M} \models p[a_1, .., a_n] \\ \frac{1}{2} & a_i = \bot \textit{ for some } i \\ 0 & \textit{if } (a_1, .., a_n) \in M^n \textit{ and } \mathcal{M} \models \neg p[a_1, .., a_n] \end{cases}
\end{array}
$$

As a consequence of this definition we get for all function symbols f in Σ that $I^3(f)(a_1, \ldots, a_n) = \bot$ if $a_i = \bot$ for some i.

Definition 18 (Three-valued Logic)
The truth value of compound formulas in a three-valued structure $\mathcal{M}^3 = (M^3 = M \cup \{\bot\}, I^3)$ will be fixed as follows:

$$
\begin{array}{lcl}
I^3(\phi_1 \wedge \phi_2) & = & min\{I^3(\phi_1), I^3(\phi_2)\} \\
I^3(\phi_1 \vee \phi_2) & = & max\{I^3(\phi_1), I^3(\phi_2)\} \\
I^3(\neg \phi) & = & 1 - I^3(\phi) \\
I^3(\phi_1 \rightarrow \phi_2) & = & I^3(\neg \phi_1 \vee \phi_2) \\
I^3(\forall x \phi) & = & min\{I^3(\phi[a]) \mid a \in M\} \\
I^3(\exists x \phi) & = & max\{I^3(\phi[a]) \mid a \in M\}
\end{array}
$$

The minimum and maxiumum operators are performed with respect to the intuitive ordering $0 < \frac{1}{2} < 1$ of the three truth values.

Note, that according to Definition 18 quantifiers only range over M and not over the whole universe $M^3 = M \cup \{\bot\}$ of $\mathcal{M}^3$.

Example 18

$$
\begin{array}{llcl}
1 & I^3(\frac{1}{0} > 0 \vee 1 > 0) & = & 1 \\
2 & I^3(\frac{1}{0} > 0 \vee \frac{1}{0} \leq 0) & = & \frac{1}{2} \\
3 & I^3(\frac{1}{0} > 0 \rightarrow 1 > 0) & = & 1 \\
4 & I^3(\forall x(\frac{1}{x} * x = x) & = & \frac{1}{2} \\
5 & I^3(\exists x(\frac{1}{x} * x = x) & = & 1
\end{array}
$$

Example 1 can be interpreted as a kind of short-cut *evaluation as known from Boolean operators in programming languages, with the difference that here the order of the disjunctive parts is not relevant. Example 2 shows that* *this short-cut evaluation does not extend to the evaluation of even simple tautologies.*

Lemma 19
Let $\mathcal{M} = (M, I)$ be a Σ-structure, t a term, ϕ a Σ-formula with the free variables $x_1, \ldots, x_n$, and $\bar{a} = a_1, \ldots, a_n$ elements in M. Then

1. *If $\mathcal{M} \models wf_t[a_1, \ldots, a_n]$ then $I(t)(a_1, \ldots, a_n) = I^3(t)(a_1, \ldots, a_n)$*
2. *$\mathcal{M} \models \mathcal{T}(\phi)[a_1, \ldots, a_n]$ iff $I^3(\phi[a_1, \ldots, a_n]) = 1$*
3. *$\mathcal{M} \models \mathcal{F}(\phi)[a_1, \ldots, a_n]$ iff $I^3(\phi[a_1, \ldots, a_n]) = 0$*
4. *$\mathcal{M} \models \mathcal{Y}(\phi)[a_1, \ldots, a_n]$ iff $I^3(\phi[a_1, \ldots, a_n]) \neq \frac{1}{2}$*

Proof
Item 1 If t is a simple term of the form $f(x_1, \ldots, x_n)$ the claim follows from Definitions 17 and 14. If t is a compound term $f(t_1, \ldots, t_n)$ the induction hypothesis, making use of the fact that $\mathcal{M} \models wf_t[\bar{a}]$ implies $\mathcal{M} \models fix_f[I^{\mathcal{M}}(t_1)[\bar{a}], \ldots, I^{\mathcal{M}}(t_n)[\bar{a}]]$ yields $I(t_i)(a_1, \ldots, a_n) = I^3(t_i)(a_1, \ldots, a_n)$ for all i. Now

$$\begin{array}{rcll} I(t)(\bar{a}) & = & I(f)(I(t_1)(\bar{a}), \ldots, I(t_n)(\bar{a}) & \text{semantics} \\ & = & I^3(f)(I(t_1)(\bar{a}), .., I(t_n)(\bar{a})) & \mathcal{M} \models fix_f[I(t_1)(\bar{a}), .., I(t_n)(\bar{a})] \\ & & & \text{and Def.17} \\ & = & I^3(f)(I^3(t_1)(\bar{a}), \ldots, I^3(t_n)(\bar{a}) & \text{Ind.Hyp} \\ & = & I^3(t)(\bar{a}) & \text{semantics} \end{array}$$

Items 2 and 3 The proof proceeds by simultaneous induction on the complexity of ϕ.
atomic formulas

$$\begin{array}{rcll} \mathcal{M} \models \mathcal{T}(p(\bar{t}))[\bar{a}] & \Leftrightarrow & \mathcal{M} \models (p(\bar{t}) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i})[\bar{a}] & \text{Def. } \mathcal{T} \\ & \Leftrightarrow & \mathcal{M} \models p[\bar{b}] \quad \text{and } b_i = I(t_i)[\bar{a}] = I^3(t_i)[\bar{a}] & \text{item 1} \\ & \Leftrightarrow & I^3(p[\bar{b}]) = 1 & \text{Def. } I^3 \\ & \Leftrightarrow & I^3(p(\bar{t})[\bar{a}]) = 1 & \text{def. of } b \\ \mathcal{M} \models \mathcal{F}(p(\bar{t}))[\bar{a}] & \Leftrightarrow & \mathcal{M} \models (\neg p(\bar{t}) \wedge \bigwedge_{i=1}^{i=n} wf_{t_i})[\bar{a}] & \text{Def. } \mathcal{F} \\ & \Leftrightarrow & \mathcal{M} \models \neg p[\bar{b}] \quad \text{and } b_i = I(t_i)[\bar{a}] = I^3(t_i)[\bar{a}] & \text{item 1} \\ & \Leftrightarrow & I^3(p[\bar{b}]) = 0 & \text{Def. } I^3 \\ & \Leftrightarrow & I^3(p(\bar{t})[\bar{a}]) = 0 & \text{def. of } b \end{array}$$

conjunction

$$
\begin{array}{rcll}
\mathcal{M} \models \mathcal{T}(\phi_1 \wedge \phi_2)[\bar{a}] & \Leftrightarrow & \mathcal{M} \models (\mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2))[\bar{a}] & \text{Def. } \mathcal{T} \\
& \Leftrightarrow & I^3(\phi_1[\bar{a}]) = 1 \text{ and } I^3(\phi_2[\bar{a}]) = 1 & \text{I.Hyp.} \\
& \Leftrightarrow & I^3((\phi_1 \wedge \phi_2[\bar{a}])) = 1 & \text{Def. } I^3 \\
\mathcal{M} \models \mathcal{F}(\phi_1 \wedge \phi_2)[\bar{a}] & \Leftrightarrow & \mathcal{M} \models (\mathcal{F}(\phi_1) \vee \mathcal{F}(\phi_2))[\bar{a}] & \text{Def. } \mathcal{F} \\
& \Leftrightarrow & \mathcal{M} \models \mathcal{F}(\phi_1)[\bar{a}] \text{ or} & \\
& & \mathcal{M} \models \mathcal{F}(\phi_2)[\bar{a}] & \vee \\
& \Leftrightarrow & I^3(\phi_1[\bar{a}]) = 0 \text{ or } I^3(\phi_2[\bar{a}]) = 0 & \text{I.Hyp.} \\
& \Leftrightarrow & I^3((\phi_1 \wedge \phi_2[\bar{a}])) = 0 & \text{Def. } I^3
\end{array}
$$

disjunction analogous to conjunction

negation

$$
\begin{array}{rcll}
\mathcal{M} \models \mathcal{T}(\neg\phi)[\bar{a}] & \Leftrightarrow & \mathcal{M} \models \mathcal{F}(\phi)[\bar{a}] & \text{Def. } \mathcal{T} \\
& \Leftrightarrow & I^3(\phi[\bar{a}]) = 0 & \text{Ind.Hyp.} \\
& \Leftrightarrow & I^3(\neg\phi[\bar{a}]) = 1 & \text{Def. } I^3
\end{array}
$$

The case $\mathcal{M} \models \mathcal{F}(\neg\phi) \Leftrightarrow I^3(\neg\phi) = 0$ follow vice versa.

quantification

$$
\begin{array}{rcll}
\mathcal{M} \models \mathcal{T}(\forall x_0\phi)[\bar{a}] & \Leftrightarrow & \mathcal{M} \models (\forall x_0 \mathcal{T}(\phi))[\bar{a}] & \text{Def. } \mathcal{T} \\
& \Leftrightarrow & \mathcal{M} \models \mathcal{T}(\phi)[a_0, \bar{a}] \text{ for all } a_0 \in M & \text{Def. } \forall \\
& \Leftrightarrow & I^3(\phi[a_0, \bar{a}]) = 1 \text{ for all } a_0 \in M & \text{Ind.Hyp.} \\
& \Leftrightarrow & I^3(\forall x_0\phi[\bar{a}]) = 1 & \text{Def. } I^3
\end{array}
$$

We leave the remaining case $\mathcal{T}(\exists x_0\phi)$, $\mathcal{F}(\forall x_0\phi)$, $\mathcal{F}(\exists x_0\phi)$ to the reader.
Item 4 follows directly from 2 and 3.

■

2.5 Examples

Example 20
Let us compute wd_ϕ for

$$
\begin{array}{l}
\phi(s_1, s_2) \equiv \\
(s_1 = s_2 \leftrightarrow seqLen(s_1) = seqLen(s_2)\ \& \\
\forall int\ i(0 \leq i < seqLen(s_1) \rightarrow any :: seqGet(s_1, i) = any :: seqGet(s_2, i)))
\end{array}
$$

This is the equality axiom SeqAxiom 1 on page 62 for the theory of sequences. It just serves as an example here and does not imply any commitment that axioms should be well-defined.

$$
\begin{array}{rcl}
wd_\phi(s_1, s_2) & = & (true \wedge true\ \&\ \forall int\ i(0 \leq i \& i < seqLen(s_1) \rightarrow \\
& & 0 \leq i\ \&\ i \leq seqLen(s_1)\ \&\ i \leq seqLen(s_2))) \\
& \leftrightarrow & (\forall int\ i(0 \leq i\ \&\ i < seqLen(s_1) \rightarrow \\
& & 0 \leq i\ \&\ i \leq seqLen(s_1)\ \&\ i \leq seqLen(s_2)))
\end{array}
$$

Further

$$\begin{array}{rcl} wd_{\forall Seq\ s_1,s_2\phi} & = & \forall Seq\ s_1, s_2 \forall int\ i(0 \leq i\ \&\ i < seqLen(s_1) \rightarrow \\ & & 0 \leq i\ \&\ i \leq seqLen(s_1)\ \&\ i \leq seqLen(s_2)) \\ & \leftrightarrow & \forall Seq\ s_1, s_2 \forall int\ i(0 \leq i\ \&\ i < seqLen(s_1) \rightarrow \\ & & i \leq seqLen(s_2)) \end{array}$$

If Th_{seq} is the overspecified theory of sequences given Section 2.7, then $Th_{seq} \not\vdash \forall s_1, s_2 wd_\phi$. But, the formula $\forall s_1, s_2\phi$ is nevertheless total. The problem is, that when computing wd for the second conjunct on the righthand side of the equivalence no use can be made of the equality $seqLen(s_1) = seqLen(s_2)$ stated in the first conjunct. This emphasizes the point the wd_ϕ is only a sufficient condition for totality and depends very much on the syntactic structure of the formula under consideration. This is in contrast with the notion of a total formula with respect to Th: If ψ is total with respect to Th and $Th \models \psi \leftrightarrow \psi'$ then also ψ' is total with respect to Th. Also the operation $\mathcal{Y}$ from Section 2.4 shares this property. $\mathcal{Y}(\phi)$ is computed in the next Example 21.

A simple syntactic variation from ϕ to an equivalent formula ϕ' would yield $Th_{seq} \vdash \forall s_1, s_2 wd_{\phi'}$:

$$\begin{array}{l} \phi'(s_1, s_2) \equiv \\ (s_1 = s_2 \leftrightarrow seqLen(s_1) = seqLen(s_2)\ \& \\ \forall int\ i(0 \leq i\ \&\ i < seqLen(s_1)\ \&\ i < seqLen(s_2) \\ \rightarrow any :: seqGet(s_1, i) = any :: seqGet(s_2, i))) \end{array}$$

Example 21
Let us compute $\mathcal{Y}(\phi)$ for the same formula as in Example 20:

$$\begin{array}{l} \phi(s_1, s_2) \equiv \\ (s_1 = s_2 \leftrightarrow seqLen(s_1) = seqLen(s_2)\ \& \\ \forall int\ i(0 \leq i < seqLen(s_1) \rightarrow any :: seqGet(s_1, i) = any :: seqGet(s_2, i))) \end{array}$$

For conciseness we write $s[i]$ for $any :: seqGet(s, i)$, Len for $seqLen$, and $\forall i$ $\exists i$ for $\forall int\ i$ $\exists int\ i$.

$$
\begin{array}{rcl}
\mathcal{T}(\phi) & = & (\mathcal{T}(s_1 \neq s_2) \ \vee \mathcal{T}(Len(s_1) = Len(s_2) \wedge \forall i(\ldots))) \wedge \\
& & (\mathcal{T}(s_1 = s_2) \ \vee \mathcal{T}(\neg(Len(s_1) = Len(s_2) \wedge \forall i(\ldots)))) \\
& = & (s_1 \neq s_2 \ \vee (Len(s_1) = Len(s_2) \wedge \mathcal{T}(\forall i(\ldots)))) \wedge \\
& & (s_1 = s_2 \ \vee (Len(s_1) \neq Len(s_2) \vee \mathcal{T}(\neg \forall i(\ldots)))) \\
& = & (s_1 \neq s_2 \ \vee (Len(s_1) = Len(s_2) \wedge \\
& & \quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow \mathcal{T}(s_1[i] = s_2[i])))) \ \wedge \\
& & (s_1 = s_2 \ \vee (Len(s_1) \neq Len(s_2) \vee \\
& & \quad \exists i(0 \leq i \wedge i < Len(s_1) \wedge \mathcal{F}(s_1[i] = s_2[i])))) \\
& = & (s_1 \neq s_2 \ \vee (Len(s_1) = Len(s_2) \wedge \\
& & \quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow \\
& & \qquad (s_1[i] = s_2[i]) \wedge 0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2)))) \ \wedge \\
& & (s_1 = s_2 \ \vee (Len(s_1) \neq Len(s_2) \vee \\
& & \quad \exists i(0 \leq i \wedge i < Len(s_1) \wedge \\
& & s_1[i] \neq s_2[i]) \wedge 0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2)) \\
& = & (s_1 = s_2 \ \rightarrow (Len(s_1) = Len(s_2) \wedge \\
& & \quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow \\
& & \qquad (s_1[i] = s_2[i]) \wedge 0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2)))) \ \wedge \\
& & s_1 \neq s_2 \rightarrow (Len(s_1) \neq Len(s_2) \vee \\
& & \exists i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \wedge s_1[i] \neq s_2[i])) \\
& = & true \ \wedge \\
& & s_1 \neq s_2 \rightarrow (Len(s_1) \neq Len(s_2) \vee \\
& & \exists i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \wedge s_1[i] \neq s_2[i]))
\end{array}
$$

$$
\begin{aligned}
\mathcal{F}(\phi) &= \mathcal{F}(s_1 \neq s_2 \ \vee (Len(s_1) = Len(s_2) \wedge \forall i(\ldots))) \vee \\
&\ \mathcal{F}(s_1 = s_2 \ \vee \neg(Len(s_1) = Len(s_2) \wedge \forall i(\ldots))) \\
&= (\mathcal{F}(s_1 \neq s_2) \wedge \\
&\ (\mathcal{F}(Len(s_1) = Len(s_2)) \vee \mathcal{F}(\forall i(\ldots)))) \ \vee \\
&\ (\mathcal{F}(s_1 = s_2) \wedge \\
&\ (\mathcal{F}(\neg(Len(s_1) = Len(s_2))) \vee \mathcal{F}(\neg\forall i(\ldots)))) \\
&= (s_1 = s_2 \wedge \\
&\quad (Len(s_1) \neq Len(s_2 \vee \\
&\quad \mathcal{F}(\forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]))))) \vee \\
&\ (s_1 \neq s_2 \wedge \\
&\quad (Len(s_1) = Len(s_2) \wedge \\
&\quad \mathcal{F}(\neg\forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i])))) \\
&= (s_1 = s_2 \wedge \\
&\quad (Len(s_1) \neq Len(s_2 \vee \\
&\quad \exists i \mathcal{F}(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i])))) \vee \\
&\ (s_1 \neq s_2 \wedge \\
&\quad (Len(s_1) = Len(s_2) \wedge \\
&\quad \mathcal{T}(\forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i])))) \\
&= (s_1 = s_2 \wedge \\
&\quad (Len(s_1) \neq Len(s_2) \vee \\
&\quad \exists i(\neg(0 \leq i) \wedge \neg(i < Len(s_1)) \wedge \mathcal{F}(s_1[i] = s_2[i])))) \vee \\
&\ (s_1 \neq s_2 \wedge \\
&\quad (Len(s_1) = Len(s_2)) \wedge \\
&\quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow \mathcal{T}(s_1[i] = s_2[i])))) \\
&= (s_1 = s_2 \wedge \\
&\quad (Len(s_1) \neq Len(s_2) \vee \\
&\quad \exists i(\neg(0 \leq i) \wedge \neg(i < Len(s_1)) \wedge \\
&\quad s_1[i] \neq s_2[i] \wedge 0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2)))) \vee \\
&\ (s_1 \neq s_2 \wedge \\
&\quad Len(s_1) = Len(s_2) \wedge \\
&\quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow \\
&\ s_1[i] = s_2[i] \wedge 0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2))) \\
&= false \vee \\
&\ (s_1 \neq s_2 \wedge \\
&\quad Len(s_1) = Len(s_2) \wedge \\
&\quad \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]))
\end{aligned}
$$

$$
\begin{array}{rcl}
\mathcal{Y}(\phi) & = & \mathcal{T}(\phi) \vee \mathcal{F}(\phi) \\
& = & (s_1 = s_2 \vee Len(s_1) \neq Len(s_2) \vee \exists i(\ldots)) \\
& & \;\vee \\
& & (s_1 \neq s_2 \wedge Len(s_1) = Len(s_2) \wedge \forall i(\ldots)) \\
& \Leftrightarrow & true\ \wedge\ true \wedge \\
& & (s_1 = s_2 \vee Len(s_1) \neq Len(s_2) \vee \exists i(\ldots) \vee \forall i(\ldots)) \\
& \Leftrightarrow & s_1 = s_2 \vee Len(s_1) \neq Len(s_2) \vee \\
& & \exists i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \wedge s_1[i] \neq s_2[i]) \vee \\
& & \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]) \\
& \Leftrightarrow & (s_1 = s_2\ \wedge (\ldots \vee \ldots)) \vee \\
& & (s_1 \neq s_2\ \wedge (\ldots \vee \ldots)) \\
& \Leftrightarrow & (s_1 = s_2\ \wedge (s_1 = s_2 \vee false \vee false \vee true)) \vee \\
& & (s_1 \neq s_2\ \wedge (false \vee Len(s_1) \neq Len(s_2) \vee \\
& & \exists i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \wedge s_1[i] \neq s_2[i]) \vee \\
& & \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]) \\
& \Leftrightarrow & s_1 = s_2 \vee \\
& & (s_1 \neq s_2\ \wedge (Len(s_1) \neq Len(s_2) \vee \\
& & \exists i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \wedge s_1[i] \neq s_2[i]) \vee \\
& & \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]))) \\
& \Leftrightarrow & s_1 = s_2 \vee \\
& & (s_1 \neq s_2\ \wedge (\\
& & \;(Len(s_1) = Len(s_2) \wedge \\
& & \;\;\forall i(0 \leq i \wedge i < Len(s_1) \wedge i < Len(s_2) \rightarrow s_1[i] = s_2[i])) \\
& & \rightarrow \\
& & \forall i(0 \leq i \wedge i < Len(s_1) \rightarrow s_1[i] = s_2[i]))) \\
& \Leftrightarrow & s_1 = s_2\ \vee (s_1 \neq s_2\ \wedge\ true) \\
& \Leftrightarrow & true
\end{array}
$$

Example 22
Let us compute wd_ϕ for

$$
\begin{array}{ll}
\phi \equiv & \\
(\forall node\ m; (m \neq start\quad -> & (\exists Seq\ s; (\text{fwpath}(s)\ \& \\
& node :: seqGet(s, 0) = start\ \& \\
& node :: seqGet(s, seqLen(s) - 1) = m\ \& \\
& pw(s) = d(m))))
\end{array}
$$

This is the first claim (1.9) of the Bellman-Ford-Lemma. We compute wd_ϕ using the fixed values formulas from Example 2.

$$
\begin{aligned}
wd_\phi \;=\; & \forall node\ m(m \neq start \rightarrow \\
& \forall Seq\ s; (\text{fwpath}(s) \rightarrow \\
& \&\ 0 \leq 0 < seqLen(s)\ \&\ node :: instance(any :: seqGet(s, 0)) \\
& \&\ 0 \leq seqLen(s) - 1 < seqLen(s)\ \& \\
& \&\ node :: instance(any :: seqGet(s, seqLen(s) - 1))\ \& \\
& \&\ \forall i(0 \leq i\ \&\ i < seqLen(s) \rightarrow \\
& edge(node :: seqGet(x_1, i), (node :: seqGet(x_1, i + 1)))) \\
& \& \\
& node :: instance(node :: seqGet(x_1, i)))
\end{aligned}
$$

Let Th_{WG} be the theory of weighted graphs presented in Subsection 4.1. Unfortunately we cannot prove $Th_{WG} \vdash wd_\phi$ since $0 < seqLen(s)$ does not follow from fwpath(s). Changing ϕ to the logically equivalent formula ϕ'

$$
\begin{aligned}
&\phi' \equiv \\
&(\forall node\ m; (m \neq start \quad - > \quad && (\exists Seq\ s; (\text{fwpath}(s)\ \&\ 0 < seqLen(s)\ \& \\
& && node :: seqGet(s, 0) = start\ \& \\
& && node :: seqGet(s, seqLen(s) - 1) = m\ \& \\
& && pw(s) = d(m))))
\end{aligned}
$$

then $Th_{WG} \vdash wd_{\phi'}$ can be proved and we know that ϕ', und thus also ϕ is a total formula.

Let us now turn to the question how to determine if two theories agree on the fixed part of their signature. We will look at two examples.

Example 23 (Weighted Graph)

The signature Σ_{WG} is as given in Definition 1 on page 2:

- **sorts** *node and int*
- **predicate** *edge* $(node, node)$
- **function** *int* $w(node, node)$

We add here that the function w is partial with the fixed values formula $fix_{w(x_1,x_2)} = edge(x_1, x_2)$.

Th^1_{WG} is given by the axioms

$$\forall node\ n; (!edge(n,n)) \tag{1.1}$$
$$\forall node\ m; (\forall node\ n; (edge(n,m) -> w(n,m) > 0)) \tag{1.2}$$

Th^2_{WG} on the other hand is given by axiom (1.1) and

$$\forall node\ m; (\forall node\ n; (w(n,m) > 0)) \tag{1.2a}$$

It is fairly obious that Th^1_{WG} and Th^2_{WG} agree on the fixed part of Σ_{WG}.

A more complex example will be studied in Section 2.6.

Example 24 (Factorial Function)
Let us start from some base theory T_0 in signature Σ_0 that contains at least the type int. As usual we consider the interpreted *semantics, i.e. in any model $\mathcal{M} = (M, I)$ of T_0 the interpretation of int is (precisely) the set of integers and the arithmetic operations $I(+)$, $I(*)$, etc have their usual mathematical meaning. The signature is extended to $\Sigma_{fact} = \Sigma_0 \cup \{fact\}$ by a unary function symbol $fact$ with fixed valued formula $fix_{fact}(x) \equiv x \geq 0$. We want to extend T_0 to axiomatize the following partial Σ_{fact}-structure $\mathcal{F}^p$:*

$$fact(n) = \begin{cases} n! & \text{if } n \geq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Here $n!$ is the factorial function. In [8, page 373] two axiomatization are presented.

T^1_{fact} adds the axioms

$$fact(0) = 1 \tag{2.4}$$
$$\forall int\ z(z > 0 \to fact(z) = z * fact(z-1)) \tag{2.5}$$

while T^2_{fact} adds the axioms

$$fact(0) = 1 \tag{2.6}$$
$$\forall int\ z(z \neq 0 \to fact(z) = z * fact(z-1)) \tag{2.7}$$

It is not hard to see that

$$\textit{For any } \Sigma_{fact}\text{-model } \mathcal{M} \textit{ of } T^1_{fact} \textit{ or } T^2_{fact} : \mathcal{M}^p_{\Sigma_{fact}} = \mathcal{F}^p \tag{2.8}$$
$$\textit{There is a (total) extention } \mathcal{M}^* \textit{ of } \mathcal{F}^p \textit{ with } \mathcal{M}^* \models T^1_{fact} \wedge T^2_{fact} \tag{2.9}$$
$$\textit{Let } \mathcal{M} \textit{ be an arbitrary (total) } \Sigma_{fact}\text{-structure, then } \mathcal{M} \models T^1_{fact}. \tag{2.10}$$

For $\mathcal{M}^* = (\mathbb{Z}, I^*)$ define $I^*(fact)(i) = 0$ for all $i < 0$. Otherwise the claims (2.8), (2.9), and (2.10) are easily verified. By Lemma 13 T^1_{fact} and T^2_{fact} agree on the fixed part of Σ_{fact}. Thus the same total formulas are derivable in both theories.

As a third possibility let T^3_{fact} be the theory obtained by adding the following axiom to T^3_{fact}.

$$\forall z(fact(z) \neq 0) \tag{2.11}$$

Then T^3_{fact} is inconsistent since a model would have to satisfy

$$\begin{array}{rcr} fact(-1) & = & (-1) * fact(-2) \\ & = & (-1) * (-2) * fact(-3) \\ & = & (-1) * (-2) * (-3) * fact(-4) \\ & = & \vdots \end{array}$$

which is impossible unless $fact(i) = 0$ for all $i < 0$.

Example 25 (Choose Operator)
Let $\Sigma_\in$ be a signature that contains the sorts $elem, set$, a constant $\emptyset$ of sort set and the binary relation $\in (elem, set)$. The $\Sigma_\in$-theory $T_\in$ consists of the axioms

$$\forall elem\ x(\neg x \in \emptyset) \tag{2.12}$$
$$\forall set\ x \forall set\ y(x = y \leftrightarrow \forall elem\ u(u \in x \leftrightarrow u \in y)) \tag{2.13}$$

Σ_{ch} extends $\Sigma_\in$ by a new function symbol $\mathrm{choose} : set \to elem$ *with the fixed value formulas $fix_{\mathrm{choose}}(x) \equiv x \neq \emptyset$. We propose two extension T^1_c and T^2_c of $T_\in$. T^1_c is obtained by adding the axiom*

$$\forall set\ s(s \neq \emptyset \to \mathrm{choose}(s) \in s) \tag{2.14}$$

T^2_c is obtained by adding the axiom

$$\forall set\ s(\mathrm{choose}(s) \in s) \tag{2.15}$$

T^2_c is inconsistent since $T^2_c \vdash \mathrm{choose}(\emptyset) \in \emptyset$ which contradicts (2.12).
One the other hand $T^1_c \vdash \mathrm{choose}(\emptyset) \notin \emptyset$ and
$T^1_c \vdash \forall set\ s(\mathrm{choose}(s) \in s \to wd_{\mathrm{choose}(s) \in s}(s))$ since $T^1_c \vdash \mathrm{choose}(s) \in s \to s \neq \emptyset$

Example 26
This example is used as a benchmark problem in [7]. The signature Σ_{bm} consists of the usual operations and relations $+$, $-$, $0, \ldots n$, $\geq$ on $\mathbb{Z}$ and one unary partial function symbol f with $fix_f(x) \equiv x \geq 0$. The following series of formulas is concidered

$$\begin{array}{lcl} \phi_0 & \equiv & f(x) = x \vee f(-x) = -x \\ \phi_n & \equiv & \phi_{n-1} \wedge (f(x+n) = x+n \vee f(-x-n) = -x-n) \end{array}$$

Well-definedness of ϕ_n is investigated with respect to the Σ_{bm}-theory T_{bm} given by the usual theory of integers as far as the signature Σ_{bm} is concerned plus the axiom $\forall x(x \geq 0) \to f(x) = x)$. Since $t_{bm} \vdash \forall x \phi_n$ all formulas are total with respect to T_{bm} by Lemma 7.

Let us compute $wd(\phi_n)$ from Definition 15. We get

$$\begin{array}{lcl} wd(\phi_0) & \Leftrightarrow & x = 0 \\ wd(\phi_1) & \Leftrightarrow & x = 0 \wedge x = 1 \\ wd(\phi_n) & \Leftrightarrow & false \text{ for all } n \geq 1 \end{array}$$

The critereon $wd(\phi_n)$ is thus too crude to prove the totality of ϕ_n.

The authors of [7] observe that $\mathcal{Y}(\phi_n)$ grows only linearly in the size of ϕ_n. One gets, e.g.,

$$\begin{array}{lll} \mathcal{Y}(\phi_0) & \Leftrightarrow_{T_{bm}} & x \geq 0 \vee -x \geq 0 \\ & \Leftrightarrow_{T_{bm}} & true \end{array}$$

Likewise $T_{bm} \vdash \mathcal{Y}(\phi_n) \leftrightarrow true$ for all n.

2.6 A Theory of Sequences

The goal of this subsection is to describe the sorted first-order theory of finite sequences that forms the theoretical basis for the taclets in Subsection 4.2. This theory will, typically, be only one part of the total theory used in an application as can be seen in Section 1.1. Let *Type* be the set of types that exist in a given application context. In particular, sorts *any* and *int* will occur in *Type*, with $\alpha \sqsubseteq any$ (α is a subtype of *any*) for all $\alpha \in Types$.

The signature Σ_{seq} of a theory of sequences contains the additional type

Seq and the following function symbols

$$\begin{array}{lcr}
\multicolumn{3}{c}{\text{Constructors}} \\
seqEmpty & : & \to Seq \\
seqSingleton & : & any \to Seq \\
seqConcat & : & Seq, Seq \to Seq \\
seqSub & : & Seq, int, int \to Seq \\
seqReverse & : & Seq \to Seq \\
\multicolumn{3}{c}{\text{Observers}} \\
alpha{::}\mathrm{seqGet} & : & Seq, int \to alpha \\
seqLen & : & Seq \to int
\end{array}$$

We did not want to include dependent types. So, we had to resort to the poor man's polymorphism. For any sort α in *Type* the signature contains a function symbols $alpha{::}\mathrm{seqGet}$. The situation is a bit ameliorated by the flexibility of the taclet mechanism that provides a schema variable for types and by the $alpha :: f$ contruct that easily allows to construct the family of symbols that derive from the polymorphic symbol f.

The following table show that partial symbols $\Sigma_{seq,p}$ of Σ_{seq}

function	fixed values formula
$seqSub(x_1, x_2, x_3)$	$0 \leq x_2 \leq x_3 < seqLen(x_1)$
$alpha{::}\mathrm{seqGet}(x_1, x_2)$	$0 \leq x_2 < seqLen(x_1) \wedge$ $alpha :: instance(any{::}\mathrm{seqGet}(x_1, x_2))$

We see that the total symbols are
$\Sigma_{seq,t} = \{seqEmpty,\ seqSingleton,\ seqConcat,\ seqReverse,\ seqLen\}$.

We introduce the theory Th_{seqC} incrementally. First Th^0_{seqC} is presented that only uses the vocabulary $\Sigma^0_{seq} = \{seqEmpty,\ seqSingleton,\ seqConcat,\ seqLen,\ int{::}\mathrm{seqGet},\ any{::}\mathrm{seqGet}\ \}$. Th^1_{seqC} extends Th^0_{seqC} by also including the symbol *seqSub* into the vocabulary, $\Sigma^1_{seq} = \Sigma^0_{seq} \cup \{seqSub\}$. Finally, Th_{seqC} is obtained as an extension of Th^2_{seqC} by also considering the remaining symbol, $\Sigma_{seq} = \Sigma^1_{seq} \cup \{seqReverse\}$.

The general approach in the theory of abstract data types is to start from a structure that fixes the semantics of the data type under consideration. We will call this structure $\mathcal{SQ}^0$, the data type of finite sequence in the signature Σ^0_{seq}.

For the purposes of this section we will restrict attention, apart from *seq*, to the sorts *int*, *any*, with *int* as subsort of *any*, $int \sqsubseteq any$. It is important to note that we exclude $seq \sqsubseteq any$. Otherwise, that would lead us to a recursive tower of types *sequences*, *sequences of sequences* and so on.

Definition 19
The structure $\mathcal{SQ}^0 = (U, I)$ *consists of*

1. *the universe* $U = \mathbb{Z} \cup A \cup Seq$ *with*
   - $\mathbb{Z}$ *as usual*
   - A *an infinite set that serves as the interpretation of the sort any*
   - $Seq = \{\langle a_0, \ldots, a_n \rangle \mid n \in \mathbb{N} \text{ and } a_i \in A\}$
2. $I(seqEmpty) = \langle\rangle$
3. $I(seqSingleton)(a) = \langle a \rangle$
4. $I(seqConcat)(\langle a_0, \ldots, a_n \rangle, \langle b_0, \ldots, b_m \rangle) = \langle a_0, \ldots, a_n, b_0, \ldots, b_m \rangle$
5. $I(seqLen)(\langle a_0, \ldots, a_n \rangle) = n + 1$
6. $I(any{::}\mathrm{seqGet})(\langle a_0, \ldots, a_n \rangle, i) = \begin{cases} a_i & \text{if } 0 \leq i \leq n \\ undef & \text{otherwise} \end{cases}$
7. $I(int{::}\mathrm{seqGet})(\langle a_0, \ldots, a_n \rangle, i) = \begin{cases} a_i & \text{if } 0 \leq i \leq n \text{ and} \\ & a_i \in \mathbb{Z} \\ undef & \text{otherwise} \end{cases}$

We want to find a first-order axiomatisation Th_{seqC} of $\mathcal{SQ}^0$. Later on we will compare the axiomatisation Th_{seqC}, with the theory Th_{seq}, and its subtheories Th^0_{seq}, Th^1_{seq}, embodied in the taclets in Section 4.2.

$$\begin{aligned}&\forall seq\ s_1 \forall seq\ s_2 (s_1 = s_2 \leftrightarrow seqLen(s_1) = seqLen(s_2)\ \&\\ &\forall int\ i(0 \leq i\ \&\ i < seqLen(s_1) \rightarrow\\ &\qquad any{::}\mathrm{seqGet}(s_1, i) = any{::}\mathrm{seqGet}(s_2, i)))\end{aligned} \tag{SeqAxiomC 1}$$

$$\forall any\ x(beta :: instance(x) \rightarrow beta{::}\mathrm{seqGet}(seqSingleton(x), 0) = x) \tag{SeqAxiomC 2}$$

$$\begin{aligned}&\forall seq\ s_1 \forall seq\ s_2 \forall int\ i(\\ &0 \leq i\ \&\ i < seqLen(s_1) \rightarrow\\ &beta{::}\mathrm{seqGet}(seqConcat(s_1, s_2), i) = beta{::}\mathrm{seqGet}(s_1, i)\\ &\wedge\\ &seqLen(s_1) \leq i\ \&\ i < seqLen(s_1) + seqLen(s_2) \rightarrow\\ &beta{::}\mathrm{seqGet}(seqConcat(s_1, s_2), i) = beta{::}\mathrm{seqGet}(s_2, i - seqLen(s_1)))\end{aligned} \tag{SeqAxiomC 3}$$

$$seqLen(seqEmpty) = 0 \tag{SeqAxiomC 4}$$

$$\forall any\ b(seqLen(seqSingleton(b)) = 1) \tag{SeqAxiomC 5}$$

$$\begin{aligned}&\forall seq\ s_1 \forall seq\ s_2(\\ &seqLen(seqConcat(s_1, s_2)) \quad = seqLen(s_1) + seqLen(s_2))\end{aligned} \tag{SeqAxiomC 6}$$

Th^0_{seqC} is given by the axioms SeqAxiomC 1 – SeqAxiomC 6.

Lemma 27
From Th^0_{seqC} associativity of the seqConcat function is derivable

$$\begin{aligned}&\forall Seq\ s_1, s_2, s_3(\\ &seqConcat(s_1, seqConcat(s_2, s_3)) = seqConcat(seqConcat(s_1, s_2), s_3)\end{aligned}$$

Proof By axiom SeqAxiomC 1 it suffices to show for each i with $0 \leq i\ \&\ i < seqLen(s_1) + seqLen(s_2) + seqLen(s_2)$ that

$$\begin{aligned}&any{::}\mathrm{seqGet}(seqConcat(s_1, seqConcat(s_2, s_3))), i)\\ &=\\ &any{::}\mathrm{seqGet}(seqConcat(seqConcat(s_1, s_2), s_3), i)\end{aligned}$$

In the following arguments axioms SeqAxiomC 3 and SeqAxiomC 6 are used repeatedly. We distinguish three cases

$\mathbf{0 \leq i \ \& \ i < seqLen(s_1)}$

$$\begin{aligned}
&any{::}\mathrm{seqGet}(seqConcat(s_1, seqConcat(s_2, s_3))), i) = \\
&any{::}\mathrm{seqGet}(s_1, i) \\
&\text{and} \\
&any{::}\mathrm{seqGet}(seqConcat(seqConcat(s_1, s_2), s_3), i) = \\
&any{::}\mathrm{seqGet}(seqConcat(s_1, s_2), i) = \\
&any{::}\mathrm{seqGet}(s_1, i)
\end{aligned}$$

$\mathbf{seqLen(s_1) \leq i \ \& \ i < seqLen(s_1) + seqLen(s_2)}$

$$\begin{aligned}
&any{::}\mathrm{seqGet}(seqConcat(s_1, seqConcat(s_2, s_3))), i) = \\
&any{::}\mathrm{seqGet}(seqConcat(s_2, s_3), i - seqLen(s_1)) = \\
&any{::}\mathrm{seqGet}(s_2, i - seqLen(s_1)) \\
&\text{and} \\
&any{::}\mathrm{seqGet}(seqConcat(seqConcat(s_1, s_2), s_3), i) = \\
&any{::}\mathrm{seqGet}(seqConcat(s_1, s_2), i) = \\
&any{::}\mathrm{seqGet}(s_2, i - seqLen(s_1))
\end{aligned}$$

$\mathbf{seqLen(s_1) + seqLen(s_2) \leq i \ \&}$
$\mathbf{i < seqLen(s_1) + seqLen(s_2) + seqLen(s_3)}$

$$\begin{aligned}
&any{::}\mathrm{seqGet}(seqConcat(s_1, seqConcat(s_2, s_3))), i) = \\
&any{::}\mathrm{seqGet}(seqConcat(s_2, s_3), i - seqLen(s_1)) = \\
&any{::}\mathrm{seqGet}(s_3, i - seqLen(s_1) - seqLen(s_2)) \\
&\text{and} \\
&any{::}\mathrm{seqGet}(seqConcat(seqConcat(s_1, s_2), s_3), i) = \\
&any{::}\mathrm{seqGet}(s_3, i - seqLen(s_1) - seqLen(s_2))
\end{aligned}$$

∎

Lemma 28
The following two formulas ϕ_i are theorems Th^0_{seqC}, i.e., we can prove $Th^0_{seqC} \vdash \phi_i$.

1. $\forall seq\ s(seqConcat(s, seqEmpty) = s)$

2. $\forall seq\ s(seqConcat(seqEmpty, s) = s)$

Proof Easy exercise using Axiom SeqAxiomC 1.

■

Definition 20

1. *A term t only built from variables x of sort any or a subsort of any and the functions seqEmpty, seqSingleton and seqConcat is called a* constructor term.
Thus a constructor term does not contain variables of type seq.

2. *A constructor term t is called a* normal form *if all subterms of t are either*
   - *seqEmtpy or*
   - *of the form $seqSingleton(x)$ for a variable x or*
   - *of the form $seqConcat(t_0, seqSingleton(x))$*

By the way part 2 is phrased it is obvious that any subterm of a normal form is also a normal form. A typical normal from thus looks like this:

$$seqConcat(seqConcat(\ldots(seqConcat(seqSingleton(x_1), seqSingleton(x_2)), \ldots), seqSingleton(x_{n-1})), seqSingleton(x_n))$$

Lemma 29
For every constructor term t there is a normal form $nf(t)$ such that

$$Th^0_{seqC} \vdash \forall \bar{x}(t = nf(t))$$

Let us first establis a bit of terminology. We call two terms t and s equivalent (in Th^0_{seqC}) if $Th^0_{seqC} \vdash \forall \bar{x}(t = nf(t))$. Using this terminology the claim of the lemma may be rephrased as: for every constructor term there is an equivalent normal form.

Proof We call a term ϵ-free if the constant *seqEmpty* does not occur in it. As a first step we show that any constructor term t is equivalent to a term t_0 such that

- $t_0 \equiv seqEmpty$ or
- t_0 is ϵ-free.

t_0 can be obtained from t by repeated application of Lemma 28.

The rough guide to the remainder of the proof is: apply the associative law from left to right as long as possible, then after finitely many step a normal from will be produced. If you are happy with that you can jump forward to the end of the proof. For the remaining audience I will spell out the, I am afraid rather laborious, details. I start be defining two integer measures of any ϵ-free constructor term

Definition 21
Let t be an ϵ-free constructor term.

1.
   - $dp(seqSingleton(x)) = 0$
   - $dp(seqConcat(t_0, t_1)) = dp(t_0) + dp(t_1) + 1$
2.
   - $dct(seqSingleton(x)) = 0$
   - $dct(seqConcat(t_0, t_1)) = dct(t_0) + dp(t_1)$

The number $dp(t)$ is just the number of occurences of the symbol $seqConcat$ in t. The number $dct(t)$ is called the *defect* of t. We first observe .18

$$\text{For all } \epsilon\text{-free constructor terms } t: \quad dct(t) \leq dp(t) \tag{2}$$

The relevant part in the inductive argument is

$$\begin{array}{rcll} dct(seqConcat(t_0, t_1)) & = & dct(t_0) + dp(t_1) & \text{Def.} \\ & \leq & dp(t_0) + dp(t_1) & \text{Ind.Hyp.} \\ & < & dp(t_0) + dp(t_1) + 1 & \text{arithmetic} \\ & = & dp(seqConcat(t_0, t_1)) & \text{Def.} \end{array}$$

It can be readily seen that $dct(t) = 0$ for any normal form t. For the initial cases in Definition 20 of a normal this is explicitly part of the Definition 21 of dct. For the inductive step case we have

$$\begin{array}{rcll} dct(seqConcat(t_0, seqSingleton(x))) & = & dct(seqSingleton(x)) & \text{Def 21} \\ & = & dct(t_0) + 0 & \text{Def 21} \\ & = & 0 & \text{Ind.Hyp} \end{array} .$$

The induction hypothesis is applicable since t_0 is again a normal form. The next claim states that also the converse is true.

$$\text{Any } \epsilon\text{-free constructor term } t \text{ with } dct(t) = 0 \text{ is a normal form.} \tag{3}$$

For the initial cases of the definition of a constructor term this is obvious. For the rest of the argument we investigate an ϵ-free constructor term t of the form $seqConcat(t_0, t_1)$ with $dct(seqConcat(t_0, t_1)) = 0$. Thus $dct(t_0) = 0$ and $dp(t_1) = 0$. The induction hypothesis tells us that t_0 is a normal form, different from *seqEmpty*. Furthermore, we know that t_1 does not contain the function symbol *seqConcat*. Since t was ϵ-free the only possibility is $t_1 \equiv seqSingleton(x)$. Our typing restriction exclude terms of the form $seqSingleton(seqSingleton(x))$. Now, it remains to observe that, if t_0 is a normal form then also $seqConcat\ (t_0,\ seqSingleton(x))$ is a normal form.

Here comes the next observation:

$$\text{For all terms } s_1, s_2, s_3 \quad \begin{array}{c} dct(seqConcat(s_1, seqConcat(s_2, s_3))) \\ > \\ dct(seqConcat(seqConcat(s_1, s_2), s_3)) \end{array} \tag{4}$$

This follows from the easy computation:

$$\begin{array}{lll} dct(seqConcat(s_1, seqConcat(s_2, s_3))) & & \\ \quad = & dct(s_1) + dp(seqConcat(s_2, s_3)) & \text{Def} \\ \quad = & dct(s_1) + dp(s_2) + dp(s_3) + 1 & \text{Def} \\ \quad > & dct(s_1) + dp(s_2) + dp(s_3)) & \text{arithm.} \\ \quad = & dct(seqConcat(s_1, s_2)) + dp(s_3)) & \text{Def} \\ \quad = & dct(seqConcat(seqConcat(s_1, s_2), s_3)) & \text{Def} \end{array}$$

We are now ready to put the pieces together to finally prove Lemma 29. Starting from an ϵ-free constructor term t we apply the rewriting rule

$$seqConcat(s_1, seqConcat(s_2, s_3)) \rightsquigarrow seqConcat(seqConcat(s_1, s_2), s_3)$$

which by Lemma 27 yields equivalent terms as long as possible. By (4) we know that this will terminate after finitely many (maybe 0) steps. The final term t_f in this series of rewriting will satisfy $dct(t_f) = 0$ any thus be a normal form by 3.

∎

Theorem 30
Let $\mathcal{M}$ be a model of Th^0_{seqC} (i.e., $\mathcal{M} \models Th^0_{seqC}$) then

$$\mathcal{M}^p_{\Sigma^0_{seq}} = \mathcal{SQ}^0,$$

where $\mathcal{M}^p_{\Sigma^o_{seq}}$ is then partial structure obtained from $\mathcal{M}$ (see Definition 9).

Proof Let $\mathcal{M} = (M, I)$. In our restricted setting $M = \mathbb{Z} \cup A \cup Seq$, with $\mathbb{Z}$, A, Seq the interpretations of the sorts *int*, *any* and *seq*. For every constructor term t and every assignment $a_1, \ldots, a_n$ of its variables $I(t)(a_1, \ldots, a_n) \in Seq$. If t is a normal form we write $\langle a_1, \ldots, a_n \rangle$ for $I(t)(a_1, \ldots, a_n) \in Seq$. This is justified since by axiom SeqAxiomC 1 $\langle a_1, \ldots, a_n \rangle = \langle b_1, \ldots, b_m \rangle$ iff $n = m$ and $a_i = b_i$ for all $0 \leq i < n$. Furthermore, for any element $s \in Seq$ we have $s = \langle a_1, \ldots, a_n \rangle$ for $n = I(seqLen)(s)$ and $a_i = I(any{::}\mathrm{seqGet})(s, i)$. Thus $Seq = \{\langle a_1, \ldots, a_n \rangle \mid n \in \mathbb{N}, a_i \in A\}$ can be identified with the interpretation of sort *seq* in the structure $\mathcal{SQ}^0$. Strictly mathematical we should define an injective and surjective function F between the universes M of $\mathcal{M}$ and U of $\mathcal{SQ}^0$ and then proceed to show that F is an isomorphism. In the remainder of the proof we will assume the F is identity. This makes notation more concise and once this simplification has been explained there is little danger of confusion. It can now be easily verified that *id* is an isomorphism, i.e., that the interpretation of the symbols *seqLen* and *alpha*::seqGet in $\mathcal{M}$ and $\mathcal{SQ}^0$ agree on the their fixed value domain. Here is a sample. We show that $I(seqLen)(\langle a_1, \ldots, a_n \rangle) = n$. Naturally, this is proved via induction on n. The cases $n = 0$ and $n = 1$ are covered by the axioms SeqAxiomC 4 nad SeqAxiomC 5. Assume that we know $I(seqLen)(\langle a_1, \ldots, a_n \rangle) = n$ and want to convince ourselves of $I(seqLen)(\langle a_1, \ldots, a_n, a_{n+1} \rangle) = n + 1$. Axiom SeqAxiomC 6 is sufficient to guarantee this. We leave it to the reader to handle the cases *int*::seqGet and *any*::seqGet.

$\blacksquare$

As a side remark we point out that for a given infinite structure $\mathcal{N}$ it is not possible to find a first order theory $T_{\mathcal{N}}$ such that for all structures $\mathcal{M}$ from $\mathcal{M} \models T_{\mathcal{N}}$ we conclude $\mathcal{N} = \mathcal{M}$. In the context of Theorem 30 it is the requirement that the interpretation of sort *int* is $\mathbb{Z}$ that goes beyond first order logic.

The property of theory Th^0_{seqC} stated in Theorem 30 is in contrast to the properties of theories Th^i_{WG} from Example 23. There may well be models $\mathcal{M}_1$, $\mathcal{M}_2$ of Th^1_{WG} such that their restrictions $(\mathcal{M}_1)^p_{\Sigma_{WG}}$, $(\mathcal{M}_2)^p_{\Sigma_{WG}}$ are different, more precisely are not isomorphic.

Lemma 31
Let $\mathcal{M}$ be a (total) Σ^0_{seq}-structure that extends $\mathcal{SQ}^0$ then

$$\mathcal{M} \models Th^0_{seqC}$$

Proof Easy.

∎

Just to give an example $\mathcal{M} = (U, I_{\mathcal{M}})$ could extend the structure $\mathcal{SQ}^0 = (U, I)$ by

$$I_{\mathcal{M}}(int :: seqGet)(\langle a_0, \ldots, a_n\rangle, i) = \begin{cases} a_i & \text{if } 0 \leq i \leq n \text{ and } a_i \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

Definition 22 **(Th^1_{seqC})**

The theory Th^1_{seqC} extends Th^0_{seqC} by including the symbol seqSub into the vocabulary, $\Sigma^1_{seq} = \Sigma^0_{seq} \cup \{seqSub\}$ and by adding the following two axioms:

$$\forall seq\ s \forall int\ i, j(i \leq j \to seqLen(seqSub(s, i, j)) = (j - i) + 1) \tag{SeqAxiomC 7}$$

$$\begin{aligned} &\forall seq\ s \forall int\ i, j, k(0 \leq i \land i \leq j \land j < seqLen(s) \land 0 \leq k \land i + k < j \land \\ &beta :: instance(seqGet(s, i + k)) \\ &\to beta :: seqGet(seqSub(s, i, j), k) = beta :: seqGet(s, i + k)) \end{aligned} \tag{SeqAxiomC 8}$$

In parallel the Σ^0_{seq}-structure $\mathcal{SQ}^0$ is expanded to a Σ^1_{seq}-structure $\mathcal{SQ}^1$

Definition 23
The structure $\mathcal{SQ}^1 = (U, I)$ coincides with $\mathcal{SQ}^0$ on the vocabulary Σ^0_{seq} and defines in addition for $s = \langle a_1, \ldots, a_n\rangle$:

$$I(seqSub)(s, i, j) = \begin{cases} s = \langle a_i, \ldots, a_j\rangle & \textit{if } 0 \leq i \leq j \leq n \\ undef & \textit{otherwise} \end{cases}$$

The analog of Theorem 30 now reads

Theorem 32
Let $\mathcal{M} = (M, I_{\mathcal{M}})$ be a model of Th^1_{seqC} then

$$\mathcal{M}^p_{\Sigma^1_{seq}} = \mathcal{SQ}^1,$$

where $\mathcal{M}^p_{\Sigma^1_{seq}}$ is then partial structure obtained from $\mathcal{M}$ (see Definition 9).

Proof Because of Theorem 30 it remains to show that the interpretations of *seqSub* agree in $\mathcal{M}$ and $\mathcal{SQ}^1$. But, this readily follows from axioms SeqAxiomC 1, SeqAxiomC 7, and SeqAxiomC 8.

∎

Lemma 33
Let $\mathcal{M}$ be a (total) Σ^1_{seq}-structure that extends $\mathcal{SQ}^1$ then

$$\mathcal{M} \models Th^1_{seqC}$$

Proof Easy.

∎

Definition 24 (Th^2_{seqC})

The theory Th^2_{seqC} extends Th^1_{seqC} by including the symbol seqReverse into the vocabulary, $\Sigma^2_{seq} = \Sigma^1_{seq} \cup \{seqReverse\}$ and by adding the following two axioms:

$$\forall seq\ s(seqLen(seqReverse(s)) = seqLen(s)) \qquad \text{(SeqAxiomC 9)}$$

$$\begin{gathered}\forall seq\ s \forall int\ k(0 \leq k \wedge k < seqLen(s) \rightarrow \\ beta :: seqGet(seqReverse(s), k) = beta :: seqGet(s, seqLen(s) - k - 1))\end{gathered} \qquad \text{(SeqAxiomC 10)}$$

In parallel the Σ^1_{seq}-structure $\mathcal{SQ}^1$ is expanded to a Σ^2_{seq}-structure $\mathcal{SQ}^2$

Definition 25
The structure $\mathcal{SQ}^2 = (U, I)$ coincides with $\mathcal{SQ}^1$ on the vocabulary Σ^0_{seq} and

$$I(seqReverse)(\langle a_1, \ldots, a_n \rangle) = \langle a_n, \ldots, a_1 \rangle$$

The analog of Theorems 30 and 32 now reads

Theorem 34
Let $\mathcal{M} = (M, I_{\mathcal{M}})$ be a model of Th^2_{seqC} then

$$\mathcal{M}^p_{\Sigma^2_{seq}} = \mathcal{SQ}^2,$$

Proof Easy. Left to the reader.

▪

Lemma 35
Let $\mathcal{M}$ be a (total) Σ^2_{seq}-structure that extends $\mathcal{SQ}^2$ then

$$\mathcal{M} \models Th^2_{seqC}$$

Proof Easy.

▪

2.7 An Overspecified Theory of Sequences

As promised before we will now inspect the theory Th_{seq} given by the taclets from Section 4.2. As a first step we translate them into the usual mathematical form.

$$\begin{aligned}
&\forall seq\ s_1 \forall seq\ s_2 (s_1 = s_2 \leftrightarrow seqLen(s_1) = seqLen(s_2)\ \& \\
&\forall int\ i (0 \leq i \& i < seqLen(s_1) \rightarrow any\!::\!\mathrm{seqGet}(s_1, i) = any\!::\!\mathrm{seqGet}(s_2, i)))
\end{aligned} \tag{SeqAxiom 1}$$

$$\begin{aligned}
&\forall alpha\ x (beta :: seqGet(seqSingleton(x), 0) = (beta)x) \\
&\wedge \\
&\forall alpha\ x \forall int\ i (i \neq 0 \rightarrow \\
&beta :: seqGet(seqSingleton(x), i) = beta :: seqGet(seqEmpty, i))
\end{aligned} \tag{SeqAxiom 2}$$

We take the axioms as they are used in the KeY system.

$$\begin{aligned}
&\forall seq\ s_1 \forall seq\ s_2 \forall int\ i (\\
&i < seqLen(s_1) \rightarrow \\
&beta :: seqGet(seqConcat(s_1, s_2), i) = beta :: seqGet(s_1, i) \\
&\wedge \\
&seqLen(s_1) \leq i \rightarrow \\
&beta :: seqGet(seqConcat(s_1, s_2), i) = beta :: seqGet(s_2, i - seqLen(s_1)))
\end{aligned} \tag{SeqAxiom 3}$$

$$seqLen(seqEmpty) = 0 \qquad \text{(SeqAxiom 4)}$$

$$\forall any\ b(seqLen(seqSingleton(b)) = 1) \qquad \text{(SeqAxiom 5)}$$

$$\begin{aligned}&\forall seq\ s_1 \forall seq\ s_2(\\ &seqLen(seqConcat(s_1, s_2)) = seqLen(s_1) + seqLen(s_2))\end{aligned} \qquad \text{(SeqAxiom 6)}$$

SeqAxiom 9 to SeqAxiom 6 make up the theory Th^0_{seq}. We observe that Axioms SeqAxiom 1, SeqAxiom 4 SeqAxiom 5 SeqAxiom 6 are the same as SeqAxiomC 1, SeqAxiomC 4 SeqAxiomC 5 SeqAxiomC 6. Furthermore it is easy to see that Axioms SeqAxiom 2 implies SeqAxiomC 2 and SeqAxiom 3 implies SeqAxiomC 3. Thus every model of Th^0_{seq} is also a model of Th^0_{seqC}.

Lemma 36
There is a model $\mathcal{M}^0_{seq}$ with $\mathcal{M}^0_{seq} \models Th^0_{seq}$

Proof by Construction We assume that only the sorts *int*, *any* are available and that $\mathbb{Z}$, A are non-empty universes interpreting them respectively with $\mathbb{Z} \subseteq A$. We assume furthermore that cast functions (int), (any) and the predicates $int :: instance$, $any :: instance$ have already been defined.

We are now ready for the construction of $\mathcal{M}^0_{seq} = (M, I)$. The universe M of $\mathcal{M}^0_{seq}$ is given by

$$\begin{array}{lcl} M & = & M_0 \cup Seq \\ M_0 & = & \mathbb{Z} \cup A \end{array}$$

with $Seq =$ the set of sequences with element from $\mathbb{Z} \cup A$. We write elements $s \in Seq$ as $s = \langle s_0, s_1, \ldots, s_{n-1}\rangle$ with $s_i \in M_0$. We use $\langle\rangle$ to stand for the empty sequence. The symbols of Σ^0_{seq} are interpreted as follows:

$$\begin{array}{lcll} I(seqEmpty) & = & \langle\rangle & \\ I(seqSingleton)(a) & = & \langle a\rangle & a \in M_0 \\ I(seqConcat)(\langle s_0, \ldots, s_{n-1}\rangle, \langle s_n, \ldots, s_{m-1}\rangle) & = & \langle s_0, \ldots, s_{m-1}\rangle & \\ I(seqLen)(\langle s_0, \ldots, s_{n-1}\rangle) & = & n & \end{array}$$

Finally

$$I(alpha{::}\mathrm{seqGet})(\langle s_0, \ldots, s_{n-1}\rangle, i) = \begin{cases} (alpha)(s_i) & \text{if } 0 \le i < n \\ 0 & \text{otherwise} \end{cases}$$

An easy inspection shows that indeed SeqAxiom 9 to SeqAxiom 6 are true in the model $\mathcal{M}^0_{seq}$ thus defined. We make use of the assumption that the cast functions and type predicates satisfy their specifications, e.g. $(int)0 = 0$ and $(any)0 = 0$.

∎

We note that Lemma 31 fails for Th^0_{seq}.

Lemma 37
Th^0_{seq} and Th^0_{seqC} agree on the fixed part of Σ^0_{seq}.

Proof The assumptions of the criterion in Lemma 13 on page 35 follow from

- Theorem 30,
- the fact that this theorem is also true for Th^0_{seq} since Th^0_{seq} is a stronger theory than Th^0_{seqC}
- Lemma 31 and
- Lemma 36

∎

The theory Th^1_{seq} extends Th^0_{seq} by including the symbol $seqSub$ into the vocabulary, $\Sigma^1_{seq} = \Sigma^0_{seq} \cup \{seqSub\}$ and by adding the following two axioms:

$$\begin{array}{l} \forall seq\ s \forall int\ i,j(\\ (i \leq j \rightarrow seqLen(seqSub(s,i,j)) = (j-i)+1) \\ \wedge \\ (i > j \rightarrow seqLen(seqSub(s,i,j)) = 0)) \end{array} \tag{SeqAxiom 7}$$

$$\begin{array}{l} \forall seq\ s \forall int\ i,j,k((0 \leq k \wedge k \leq j-i \rightarrow \\ beta :: seqGet(seqSub(s,i,j),k) = beta :: seqGet(s,i+k)) \wedge \\ k < 0 \vee k < j-i \rightarrow \\ beta :: seqGet(seqSub(s,i,j),k) = beta :: seqGet(seqEmpty,0))) \end{array} \tag{SeqAxiom 8}$$

It can be easily seen that (SeqAxiom 7 $\rightarrow$ SeqAxiomC 7) and (SeqAxiom 8 $\rightarrow$ SeqAxiomC 8) are tautologies. This Th^1_{seq} is a stronger theory than Th^1_{seqC}. This proves

Theorem 38
Let $\mathcal{M}$ be a model of Th^1_{seq} then

$$\mathcal{M}^p_{\Sigma^1_{seq}} = \mathcal{SQ}^1,$$

Proof Follows from Theorem 32 and the remarks preceeding the statement of this theorem.

■

Axioms SeqAxiom 7 and SeqAxiom 8 are typical examples of what is occasionally called the overspecification in underspecification. We observe e.g., $Th^1_{seq} \vdash \forall seq\ s(seqLen(seqSub(s,1,0) = 0)$ while this formula is not derivable in Th^1_{seqC}. Or $Th^1_{seq} \vdash \forall seq\ s(any :: seqGet(seqSub(s,0,1),2) = any :: seqGet(s,2))$ and this formula is also not derivable in Th^1_{seqC}. The second example seems to be even more serious, since on the lefthand side the sequence $seqSub(s,0,1)$ is certainly undefined at position 2, however the righthand side is defined if $seqLen(s) \geq 3$. Yet, both formulas are not total. We will in fact find out that Th^1_{seq} and Th^1_{seqC} agree on total formulas.

Lemma 39
There is a model $\mathcal{M}^1_{seq}$ with $\mathcal{M}^1_{seq} \models Th^1_{seq}$

Proof by Construction Let $\mathcal{M}^0_{seq} = (M, I^0)$ be the (total) model satisfying $\mathcal{M}^0_{seq} \models Th^0_{seq}$, already mentioned in passing after the proof of Lemma 36 on page 63 satisfying

$$I^0(int :: seqGet)(\langle a_0,\ldots,a_n\rangle, i) = \begin{cases} a_i & \text{if } 0 \leq i \leq n \text{ and } a_i \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{M}^1_{seq} = (M, I^1)$ is obtained from $\mathcal{M}^0_{seq}$ by defining $I^1(seqSub)$ and let I^1 agree with I^0 on signature Σ^0_{seq}.

$$I^1(seqSub)(\langle a_0,\ldots,a_{n-1}\rangle, i, j) = \begin{cases} seqEmpty & \text{if } j > i \\ \langle a_i,\ldots,a_j\rangle & \text{if } i \leq j \\ \text{with} & \\ a_i = I^0(any : seqGet(\langle a_0,\ldots,a_{n-1}\rangle, i) & \end{cases}$$

Obviously SeqAxiom 7 is satisfied.

We now turn to argue that $\mathcal{M}^1_{seq} \models$SeqAxiom 8. So let $s = \langle a_0,\ldots,a_{n-1}\rangle$ be an element in M_{seq} and i,j,k be given with $0 \leq k \wedge k \leq (j-i)$.
If $j < i$ we need to verifiy

$$
\begin{aligned}
I^1(beta :: seqGet(seqSub(s,i,j)))(k) &= I^0(beta :: seqGet(seqEmpty))(k)\\
&= 0\\
&= I^0(beta :: seqGet(seqEmpty))(0)\\
&= I^1(beta :: seqGet(seqEmpty))(0)
\end{aligned}
$$

If $j \geq i$ we get by definition $I^1(seqSub(s,i,j)) = \langle a_i, \ldots, a_j \rangle$ and thus

$$
\begin{aligned}
I^1(beta :: seqGet(seqSub(s,i,j)))(k) &= I^0(beta :: seqGet(\langle a_i, \ldots, a_j \rangle))(k)\\
&= I^0(beta :: seqGet(\langle a_0, \ldots, a_{n-1} \rangle))(i+k)\\
&= I^1(beta :: seqGet(\langle a_0, \ldots, a_{n-1} \rangle))(i+k)
\end{aligned}
$$

∎

The axioms SeqAxiom 7 and SeqAxiom 8 have been chosen to be as simple as possible neclecting the fact that they may entail strange looking consequence. This is safe because of the following lemma

Lemma 40
Th^1_{seq} and Th^1_{seqC} agree on the fixed part of Σ^1_{seq}.

Proof The assumptions of the criterion in Lemma 13 on page 35 follow from

- Theorem 32,
- the fact that this theorem is also true for Th^1_{seq} since Th^1_{seq} is a stronger theory than Th^1_{seqC}
- Lemma 33 and
- Lemma 39

∎

One has to be careful however not to chose that axioms simpler than possible. If we had instead of SeqAxiom 8 adopted the even more liberal axiom $SeqAxiom(*)$

$$
\begin{aligned}
&\forall seq\ s \forall int\ i,j,k((i \leq j \rightarrow\\
&beta :: seqGet(seqSub(s,i,j),k) = beta :: seqGet(s,i+k)) \wedge\\
&i > j \rightarrow\\
&beta :: seqGet(seqSub(s,i,j),k) = beta :: seqGet(seqEmpty,0)))
\end{aligned}
$$

the resulting theory Th^*_{seq} would have been inconsistent. To see this let $s_0 = \langle 0,0,0,0 \rangle$ as a shorthand for

$$
\begin{aligned}
s_0 \;=\; & seqConcat(seqConcat(seqConcat(seqSingleton(0),\\
& seqSingleton(0)), seqSingleton(0)), seqSingleton(0))
\end{aligned}
$$

Similarly let s_1 denote $\langle 1,1,1,1\rangle$. From $SeqAxiom(*)$ we obtain

$$
\begin{array}{lcl}
int :: seqGet(seqSub(s_0,1,0),2) & = & int :: seqGet(s_0,3) = 0 \\
\text{and} & & \\
int :: seqGet(seqSub(s_1,1,0),2) & = & int :: seqGet(s_1,3) = 1
\end{array} .
$$

But, $seqSub(s_0,1,0) = seqEmpty = seqSub(s_1,1,0)$, thus

$$
\begin{aligned}
0 &= int :: seqGet(seqSub(s_0,1,0),2)\\
&= int :: seqGet(seqEmpty,3)\\
&= int :: seqGet(seqSub(s_1,1,0),2)\\
&= 1
\end{aligned}
$$

The operation *seqReverse* has no well-definedness restrictions, so the extension of the above results to the theory Th^2_{seq} offers no suprise and is left to the reader.

2.8 Closing Remarks

For pratical purposes it is essential to have a series of lemmata available, useful and frequently used consequences of the axioms.

An extremely valuable instrument of Th^0_{seq} is the (structural) induction axiom. If a formula is true of the empty sequence and whenever it is true of a sequence s then it is also true for all sequences that are one entry longer than s than the formula is true for all sequences.

$$
\begin{aligned}
&(\phi[seqEmtpy/s]\wedge\\
&\forall seq\ s \forall any\ b(\phi \to \phi[seqConcat(s, seqSingleton(b))/s]))\\
&\to \forall seq\ s\phi[s])
\end{aligned}
\tag{SeqAxiom 9}
$$

For most pratical purposes this is too crude and you might want to use

$$
\begin{aligned}
&(\phi[seqEmtpy/s]\wedge\\
&\forall seq\ s \forall alpha\ b(\phi \wedge alpha :: sequence(s) \to\\
&\qquad\qquad \phi[seqConcat(s, seqSingleton(b))/s]))\\
&\to \forall seq\ s(alpha :: sequence(s) \to \phi[s])
\end{aligned}
\tag{SeqAxiom 9a}
$$

where

$$
\begin{array}{rcl}
alpha :: sequence(s) & \equiv & s = seqEmpty \vee \\
& & \forall int\ i((0 \leq i \wedge i < seqLen(s)) \rightarrow \\
& & alpha :: instance(any{::}\mathrm{seqGet}(s, i)))
\end{array}
$$

Section 4.2 also contains a couple of useful taclets, whose correctness can be derived from the axioms SeqAxiom 1 to SeqAxiom 8 plus the defining axiom for the *reverse* operation.

Chapter 3

Conservative Extension

3.1 Review of Basic Definitions

The assumptions used in the proof as described in Section 1.4 are not all of the same kind.

- (1.1), (1.2)
  are the axioms of weighted graphs,
- (1.5), (1.6)
  are assumptions on the function d that plays a centrol role in the algorithm,
- (1.7), (1.8)
  formalize the Bellman-Ford equations,
- (1.3), (1.4)
  definitions of the concepts of a path and path weight used in the claim of the lemma,
- (1.11)
  Definition of d-path

The last item is different from the rest. The new function symbol $fdpath$ defined there is completely auxiliary. It does not occur in neither claim. So we expect that also the implications

$$(1.1)\ \&\ (1.2)\ \&(1.3)\ \&\ (1.4)\ \&(1.5)\ \&\ (1.6)\ \&\ (1.7)\ \&\ (1.8)\ \rightarrow\ (1.9)$$
$$(1.1)\ \&\ (1.2)\ \&(1.3)\ \&\ (1.4)\ \&(1.5)\ \&\ (1.6)\ \&\ (1.7)\ \&\ (1.8)\ \rightarrow\ (1.10)$$

without (1.11) are universally valid. This section will provide the necessary concepts and theorems to show that this is indeed the case.

We start with the relevant definition

Definition 26 (Conservative Extension)

Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i set of sentences in Fml_{Σ_i}. T_1 is called a conservative extension *of T_0 if for all sentences $\phi \in Fml_{\Sigma_0}$:*

$$T_0 \vdash \phi \Leftrightarrow T_1 \vdash \phi$$

This definition in particular entails that a conservative extension of a consistent theory is again consistent.

Note on terminology: frequently we use the term *theory* just as a synonym for a set of formulas without free variables. Strictly speaking a theory is a set of formulas without free variables that is closed under logical inference, i.e., for a theory T in the strict sense $\{\phi \mid T \vdash \phi\} = T$. A set of axioms A axiomatizes a theory T if $\{\phi \mid A \vdash \phi\} = T$.

In our situation let $T_0^{BF} = \{$ (1.1), (1.2), (1.3), (1.4) & (1.5), (1.6), (1.7), (1.8) $\}$ and $T_1^{BF} = T_0^{BF} \cup \{(1.11)\}$. Is T_1^{BF} a conservative extension of T_0^{BF}? Fortunately, there is a simple semantical criterion to answer this question. We need a simple observation and some terminology.

Lemma 41 (Coincidence Lemma)

Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and $\phi \in Fml_{\Sigma_0}$. Furthermore let $\mathcal{M}_0$ be a Σ_0-structure and $\mathcal{M}_1$ an Σ_1-expansion of $\mathcal{M}_0$. Then

$$\mathcal{M}_0 \models \phi \quad \Leftrightarrow \quad \mathcal{M}_1 \models \phi$$

Proof Obvious.

∎

Definition 27 (Expansion)
Let $\Sigma_0 \subseteq \Sigma_1$ be signatures.

1. *A Σ_1-structure $\mathcal{M}_1 = (M_1, I_1)$ is called an* expansion *of a Σ_0-structure $\mathcal{M}_0 = (M_0, I_0)$ if*

   - *$M_0 = M_1$ and*

- *for all* $f, p \in \Sigma_0$ $I_1(f) = I_0(f)$ *and* $I_1(p) = I_0(p)$.

2. *If* $\mathcal{M}_1 = (M_1, I_1)$ *is a* Σ_1*-structure the structure obtained from it by omitting the interpretations of all symbols in* $\Sigma_1 \setminus \Sigma_0$ *is called the* restriction *of* $\mathcal{M}_1$ *to* Σ_0 *and denoted by* $(\mathcal{M}_1 \upharpoonright \Sigma_0)$

Definition 28 (Semantic Conservative Extension)

Let $\Sigma_0 \subseteq \Sigma_1$ *be signatures, and* T_i *sets of sentences in* Fml_{Σ_i}.
T_1 *is called a* semantic conservative extension *of* T_0 *if*

1. *for all* Σ_1*-structures* $\mathcal{M}_1$

$$\mathcal{M}_1 \models T_1 \Rightarrow (\mathcal{M}_1 \upharpoonright \Sigma_0) \models T_0$$

2. *for every* Σ_0*-structure* $\mathcal{M}_0$ *with* $\mathcal{M}_0 \models T_0$ *there is a* Σ_1*-expansion* $\mathcal{M}_1$ *of* $\mathcal{M}_0$ *with* $\mathcal{M}_1 \models T_1$.

Theorem 42 (Criterion for Conservative Extension)

Let $\Sigma_0 \subseteq \Sigma_1$ *be signatures, and* T_i *sets of sentences in* Fml_{Σ_i}.
If T_1 *is a semantic conservative extension of* T_0 *then it is also a (syntactic) conservative extension.*

Proof Let ϕ be a sentence in Fml_{Σ_0} with $T_0 \vdash \phi$. Let $\mathcal{M}_1$ be an arbitrary Σ_1-structure. By assumption $(\mathcal{M}_1 \upharpoonright \Sigma_0) \models T_0$. Thus we also have $(\mathcal{M}_1 \upharpoonright \Sigma_0) \models \phi$. By the coincidence lemma 41 we also have $\mathcal{M}_1 \models \phi$. In total we have shown $T_1 \vdash \phi$.
Now, assume $T_1 \vdash \phi$. If $\mathcal{M}_0$ is an arbitrary Σ_0-structure there is by the assumption an expansion of $\mathcal{M}_0$ to a Σ_1-structure $\mathcal{M}_1$. From $T_1 \vdash \phi$ we thus get $\mathcal{M}_1 \models \phi$. The coincidence lemma 41 tells us again that also $\mathcal{M}_0 \models \phi$. In total we arrive at $T_o \vdash \phi$.

■

Lemma 43 (Definitional Extension)

Let T_0 *be a set of sentences in the signature* Σ_0. *Let* $\Sigma_1 = \Sigma_0 \cup \{f\}$ *for a new* n*-place function symbol* f.

Let t *be a term in* Σ_0 *with at most the variables* $v_1, \ldots, v_n$. *Then*

$$T_1 = T_0 \cup \{\forall v_1 \ldots \forall v_n (f(v_1, \ldots, v_n) = t\})$$

is a conservative extension of T_0.

If T_1 arises in this way from T_0 we say that T_1 is a *definitional extension* of T_0.

Proof It suffices to show that T_1 is a semantic conservative extension of T_0. So let $\mathcal{M}_0 = (M, I_0)$ be a model of T_0. The expansion $\mathcal{M}_1 = (M, I_1)$ of $\mathcal{M}_0$ with $\mathcal{M}_1 \models T_1$ can simply be defined by

$$I_1(f)(m_1, \ldots, m_n) = t^{(\mathcal{M}_0.\beta)}$$

with $\beta(v_i) = m_i$.

∎

Our question at the beginning of this section is now answered since T_1^{BF} is a definitional extension of T_0^{BF}.

Lemma 44 (Definable Extension)

Let T_0 *be a set of sentences in the signature* Σ_0. *Let* $\Sigma_1 = \Sigma_0 \cup \{f\}$ *for a new* n*-place function symbol* f *and let* ϕ *be a* Σ_0 *formula with at most the free variables* $v, v_1, \ldots, v_n$.

Assume that $T_0 \vdash \forall v_1 \ldots \forall v_n \exists v(\phi(v, v_1, \ldots, v_n))$

Then

$$T_1 = T_0 \cup \{\forall v_1 \ldots \forall v_n \forall v(f(v_1, \ldots, v_n) = v \leftrightarrow \phi\})$$

is a conservative extension of T_0.

Proof Follows the lines of the proof of Lemma 43.

∎

Lemma 45 (Skolem Extensions)
Let ψ *be a formula with free variables* $w_0, w_1, \ldots, w_n$ *in the signature* Σ *and* T *a* Σ*-theory. The Skolem extension* $T_{\exists w_0 \psi}$ *of* T *for* $\exists w_0 \psi$ *is obtained by*

1. *adding an* n*-place function symbols* f *to* Σ *(sometimes denoted in greater detail as* $f_{\exists w_0 \psi}$*) and*

2. *adding the axiom*

$$\forall w_1, \ldots, w_n(\exists w_0 \psi \to \psi(f(w_1, \ldots, w_n), w_1, \ldots, w_n))$$

to T

$T_{\exists w_0 \psi}$ is a conservative extension of T.

Let us remark that Lemma 45 is a special case of Lemma 44. Let ψ be the formula addressed in Lemma 45 with free variables w, w_1, …, w_n. Use Lemma 44 for the formula

$$\phi(w, w_1, \ldots, w_n) \equiv (\exists w \psi) \to \psi.$$

Now, $\forall w_1, \ldots, w_n \exists w \phi$ obviously is a tautology. Since Skolem extensions are such an ubiquitously occuring special case we decided to devote a seperate lemma to them.

Proof We argue that $T_{\exists w_0 \psi}$ is a semantic conservative extension of T. So let $\mathcal{M} = (M, I)$ be an arbitrary Σ-model of T. We define for $a_1, \ldots, a_n = \bar{a} \in M$

$$I(f)(\bar{a}) = \begin{cases} \text{some } a \quad \text{with } \mathcal{M} \models \psi[a, \bar{a}] & \text{if } \mathcal{M} \models \exists w_0 \psi[\bar{a}] \\ \text{arbitary} & \text{if } \mathcal{M} \models \neg \exists w_0 \psi[\bar{a}] \end{cases}$$

It is easy to see that this way we obtain a model of $T_{\exists w_0 \psi}$.

∎

You can think of other variants of Lemmas 43 and 44, e.g., in theories where an induction principle is available the definition of a function could use recursion, etc.

Example 46
Here finally is an example of a theory extension that is not conservative. Let T_{DG} be the theory of directed graphs. The vocabulary Σ_{DG} consists of one binary relation $N(x, y)$. If $\mathcal{G} = (G, I)$ is a Σ_{DG} structure, then $I(N)(g_1, g_2)$ for $g_1, g_2 \in G$ models the fact that there is a directed edge from node g_1 to g_2. We do not impose any restriction on the edge relation. T_{DG} thus has no axioms. We regard equality $=$ as a logical symbol, and the axioms for equality as logical axioms. These, of course, are available for derivations in the theory T_{DG}.

The theory of ordered directed graphs T_{ODG} contains one more binary relation symbol, $\Sigma_{ODG} = \{N, <\}$ and the following axioms

$\forall x(\neg x < x)$	*strictness*
$\forall x, y, z(x < y \wedge y < z \to x < z$	*transitivity*
$\forall x, y(N(x, y) \to x < y)$	*compatibility*

Thus $<$ is a strict order relation that is compatible with the edge relation

N. If $\mathcal{G} = (G, I)$ is a model of T_{ODG} the relation $I(<)$ is frequently called a topological odering of the graph $(G, I(N))$. It is well-known that only acyclic graphs can be topologically ordered. Thus T_{ODG} is not a semantic conservative extension of T_{DG}. It can also be easily seen that T_{ODG} is not a (syntactic) conservative extension of T_{DG}:
$T_{ODG} \vdash \forall x \neg N(x,x)$ *and* $T_{ODG} \vdash \neg \exists x, y, z (N(x,y) \wedge N(y,z) \wedge N(y,z))$
but $T_{DG} \not\vdash \forall x \neg N(x,x)$ *and* $T_{DG} \not\vdash \neg \exists x, y, z (N(x,y) \wedge N(y,z) \wedge N(y,z))$

3.2 Digression

The reverse of Theorem 42 is also true in a restricted context. Before we state the precise claim we start with a preparation, the definition and relevance of the diagram $Diag(\mathcal{M})$ of a structure $\mathcal{M}$.

Definition 29
Let $\mathcal{M}$ be a Σ-structure.
The signature Σ_M is obtained from Σ by adding new constant symbols c_a for every element $a \in M$.

The expansion of $\mathcal{M}$ to a Σ_M-structure $\mathcal{M}^* = (M, I^*)$ is effected by the obvious $I^*(c_a) = a$.

Definition 30 (Diagram of a structure)
Let $\mathcal{M}$ be a Σ-structure. The diagram of $\mathcal{M}$, in symbols $Diag(\mathcal{M})$, is defined by

$$Diag(\mathcal{M}) = \{\phi \in Fml_{\Sigma_M} \mid \mathcal{M}^* \models \phi \text{ and } \phi \text{ is quantifierfree}\}$$

Lemma 47
Let $\mathcal{M}$ be a Σ-structure.
If $\mathcal{N} \models Diag(\mathcal{M})$ then $\mathcal{M}$ is (isomorphic to) a substructure of $\mathcal{N}$.

Proof Easy.

∎

Furthermore, we will need the following observation:

Lemma 48
Let $\mathcal{M}_0$ be a substructure of $\mathcal{M}$ and ϕ logically equivalent to a universal sentence. Then

$$\mathcal{M} \models \phi \Rightarrow \mathcal{M}_0 \models \phi$$

Proof Easy induction on the complexity of ϕ.

∎

To make this notes as self-contained as possible we also repeat the definition of a substructure here:

Definition 31 (Substructure)
Let $\mathcal{M} = (M, I)$ and $\mathcal{M}_0 = (M_0, I_0)$ be Σ-structures.
$\mathcal{M}_0$ is called a substructure *of $\mathcal{M}$ iff*

1. *$M_o \subseteq M$*

2. *for every n-ary function symbol $f \in \Sigma$ and any n of elements $a_1, \ldots, a_n \in M_0$*

$$I(f)(a_1, \ldots, a_n) = I_0(f)(a_1, \ldots, a_n)$$

3. *for every n-ary relation symbol $p \in \Sigma$ and any n of elements $a_1, \ldots, a_n \in M_0$*

$$(a_1, \ldots, a_n) \in I(p) = (a_1, \ldots, a_n) \in I_0(p)$$

We are now ready to state and proof the main result of this subsection.

Lemma 49
Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i sets of sentences in Fml_{Σ_i} and assume that

1. *T_1 contains only universal sentences and*

2. *$\Sigma_1 \setminus \Sigma_0$ contains only relation symbols.*

If T_1 is a conservative extension of T_0
then T_1 is also a semantic conservative extension of T_0

Proof We need to show the two clauses in Definition 28.
(1): Let $\mathcal{M}_1$ be a Σ_1-structure with $\mathcal{M}_1 \models T_1$ and $\mathcal{M}_0$ its restriction to Σ_0, i.e., $\mathcal{M}_0 = \mathcal{M}_1 \restriction \Sigma_0$. For all $\phi \in T_0$ obviously $T_0 \vdash \phi$. Thus also $T_1 \vdash \phi$ and therefore $\mathcal{M}_1 \models \phi$. By the coincidence lemma this gives $\mathcal{M}_0 \models \phi$. Thus, we get $\mathcal{M}_0 \models T_0$ as desired.
(2): Here we look at a Σ_0-structure $\mathcal{M}_0$ with $\mathcal{M}_0 \models T_0$. We set out

to find an expansion $\mathcal{M}_1$ of $\mathcal{M}_0$ with $\mathcal{M}_1 \models T_1$. To this end we consider the theory $T_1 \cup Diag(\mathcal{M}_0)$. If this theory were inconistent than already $T_1 \cup F$ for a finite subset $F \subseteq Diag(\mathcal{M}_0)$ would be inconsistent. This is the same as saying $T_1 \vdash \neg F$. Since the constants c_a do not occur in T_1 we get furthermore $T_1 \vdash \forall x_1, \ldots, x_n \neg F'$, where F' is obtained from F be replacing all occurences of constants c_a by the same variable x_i. This is equivalent to $T_1 \vdash \neg\exists x_1, \ldots, x_n F'$. Since T_1 was assume to be a conservative extension of T_0 we also get $T_0 \vdash \neg\exists x_1, \ldots, x_n F'$ and thus $\mathcal{M}_0 \models \neg\exists x_1, \ldots, x_n F'$. This is a contradiction since by the definition of $Diag(\mathcal{M}_0)$ we have $\mathcal{M}_0 \models \exists x_1, \ldots, x_n F'$ by instantiating the quantified variable x_i that replaces the constant c_a by the element a. This contradiction shows that $T_1 \cup Diag(\mathcal{M}_0)$ is consistent. Let $\mathcal{N}$ be a model of this theory. By Lemma 47 we may assume that $\mathcal{M}_0$ is a substructure of $(\mathcal{N} \upharpoonright \Sigma_0)$. Since by assumption only new relation symbols are added when passing from Σ_0 to Σ_1 also $(\mathcal{N} \upharpoonright \Sigma_1)$ is a substructure of $\mathcal{N}$. By Lemma 48 we get $(\mathcal{N} \upharpoonright \Sigma_1) \models T_1$. Obviously, $(\mathcal{N} \upharpoonright \Sigma_1)$ is an expansion of $(\mathcal{N} \upharpoonright \Sigma_0) = \mathcal{M}_0$ and we are finished. ▪

We note that the proof of Lemma 49 make use of the completeness theorem of first-order predicate logic. More precisely, the completeness theorem is used in the form:

If a set Γ of formulas is consistent, then it has a model.

As a consequence this proof does not work in the context of first-order logic with interpreted symbols, see e.g., [18, Subsection 4.2].

Lemma 49 is not true without restrictions. The following example from [11] shows that there are theories T_0 and T_1 such that T_1 is a conservative extension of T_0 but not a semantic conservative extension of T_0

Example 50
Let $\Sigma_0 = \{R, f, 0\}$ *be the signature with*

- *a binary relation symbol* R
- *a unary function symbol* f
- *a constant symbol* 0

Let $\Sigma_1 = \Sigma_0 \cup \{B, \omega\}$ *with*

- *a unary relation symbol* B

- *a constant symbol ω*

Theory T_0 is given by the axioms

1. $\forall x R(x, f(x))$
2. $\forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$
3. $\forall x \neg R(x,x)$

T_1 is T_0 plus the following axioms

4. $B(0)$
5. $\forall x (B(x) \rightarrow B(f(x)))$
6. $\neg B(\omega)$

T_1 is a conservative extension of T_0
Obviously, $T_0 \vdash \phi$ implies $T_1 \vdash \phi$ for every Σ_0-formula ϕ.
Now assume $T_1 \vdash \phi$. We want to show $T_0 \vdash \phi$. Assume this is not the case. Thus there is a Σ_0-model $\mathcal{M}$ of $T_0 \cup \{\neg\phi\}$. The axioms of T_0 imply that for all $n \in \mathbb{N}$ we have $\mathcal{M} \models R(f^n(0), f^{n+1}(0))$ and the elements $0^{\mathcal{M}}$, $f(0)^{\mathcal{M}}$, $\ldots f^n(0)^{\mathcal{M}}$ are all different. In particular $\mathcal{M} \models R(f^i(0), f^n(0))$ for all $0 \leq i < n$. This shows that any finite subset of $T_0 \cup \{\neg\phi\} \cup \{R(f^n(0), \omega) \mid n \in \mathbb{N}\}$ is satisfiable. By the compactness theorem $T_0 \cup \{\neg\phi\} \cup \{R(f^n(0), \omega) \mid n \in \mathbb{N}\}$ is satisfiable, say by $\mathcal{N}$. If we define $B^{\mathcal{N}_1} = \{f^i(0)^{\mathcal{N}} \mid i \in \mathbb{N}\}$ then $\mathcal{N}_1$ satisfies axiom 4. and 5. of T_1. If $\mathcal{N} \models \omega = f^n(0)$ we would have $\mathcal{N} \models R(\omega, \omega)$ which contradicts axiom 3 of T_0. This $\mathcal{N}_1$ is a model of $T_1 \cup \{\neg\phi\}$ contradicting our assumption. This completes the proof by contradiction of $T_0 \vdash \phi$.

T_1 is not a conservative semantic extension of T_0
Let $\mathcal{M}$ by a structure with universe $M = \mathbb{N}$, $0^{\mathcal{M}} = 0$, $f^{\mathcal{M}}(n) = n+1$ and $R^{\mathcal{M}}(n,m) \Leftrightarrow n < m$. It can be easily checked that $\mathcal{M} \models T_0$. If $\mathcal{M}_1$ were an Σ_1-expansion of $\mathcal{M}$ with $\mathcal{M}_1 \models T_1$ axiom 5 entails $B^{\mathcal{M}_1} = \mathbb{N} = M$. Thus axiom 6 cannot be satisfied.

We note that this example violates assumption 2 of Lemma 49 since $\Sigma_1 \setminus \Sigma_0$ also contains a constant symbol.

Chapter 4

Taclets

4.1 Taclets for some Axioms

Taclet for Unfolding the Definition of fwpath

```
fwPath{
 \schemaVar  \term Seq seq;
 \schemaVar \variables int iv;
 \find(fwpath(seq))
 \varcond(\notFreeIn(iv, seq))
 \replacewith(\forall iv;((0<=iv & iv<seqLen(seq)-1) ->
   edge(node::seqGet(seq,iv),node::seqGet(seq,iv+1))))
};
```

Taclet Forward Structural Seq Induction

```
seqInd_forward{
   \varcond (\notFreeIn (m,b))
   "Base Case": \add ( ==> {\subst s; seqEmpty}(b) );
   "Step Case": \add ( ==> \forall s;(\forall m ;
    (b->{\subst s;(seqConcat(s,seqSingleton(m)))}b) ));
   "Use Case":  \add ( \forall s;(b) ==>)
};
```

Taclet Backward Structural Seq Induction

```
seqInd_backward{
  \varcond (\notFreeIn (m,b))
  "Base Case": \add ( ==> {\subst s; seqEmpty}(b) );
  "Step Case": \add ( ==> \forall s;(\forall m ;
   (b->{\subst s;(seqConcat(seqSingleton(m),s))}b) ));
  "Use Case":  \add ( \forall s;(b) ==>)
};
```

Taclet for Unfolding the Path Weight Function

```
pwSum{\schemaVar  \term Seq seq;
  \schemaVar  \variables int uSub;
  \find(pw(seq))
  \varcond(\notFreeIn(uSub,seq))
  \replacewith(bsum{uSub;} (0, seqLen(seq)-1,
  w(node::seqGet(seq,uSub),node::seqGet(seq,uSub+1))))
};
```

Taclet for Unfolding the Definition of a *fdpath*

```
fdPath{ \schemaVar  \term Seq seq;
 \schemaVar \variables int iv;
 \find(fdpath(seq))
 \varcond(\notFreeIn(iv, seq))
 \replacewith( fwpath(seq) &
  \forall iv;((0<=iv & iv<seqLen(seq)-1) ->
    d(node::seqGet(seq,iv+1)) =
      d(node::seqGet(seq,iv)) +
      w(node::seqGet(seq,iv),node::seqGet(seq,iv+1))))
};
```

4.2 Taclets for the Sequence Data Type

```
\sorts {
    Seq;
}
```

```
\functions {
    //getters
    alpha alpha::seqGet(Seq, int);
    int seqLen(Seq);

    //constructors
    Seq seqEmpty;
    Seq seqSingleton(any);
    Seq seqConcat(Seq, Seq);
    Seq seqSub(Seq, int, int);
    Seq seqReverse(Seq);
}

\rules {

    //---------------------------------------------------------------
    //axioms
    //---------------------------------------------------------------

    getOfSeqSingleton {
        \schemaVar \term alpha x;
        \schemaVar \term int idx;

        \find(beta::seqGet(seqSingleton(x), idx))

        \replacewith(\if(idx = 0)
                     \then((beta)x)
                     \else(beta::seqGet(seqEmpty, 0)))

        \heuristics(simplify)
    };

    getOfSeqConcat {
        \schemaVar \term Seq seq, seq2;
        \schemaVar \term int idx;

        \find(beta::seqGet(seqConcat(seq, seq2), idx))
```

```
      \replacewith(
       \if(idx < seqLen(seq))
       \then(beta::seqGet(seq, idx))
       \else(beta::seqGet(seq2, idx - seqLen(seq))))

      \heuristics(simplify_enlarging)
  };

getOfSeqSub {
 \schemaVar \term Seq seq;
 \schemaVar \term int idx, from, to;

 \find(beta::seqGet(seqSub(seq, from, to), idx))

  \replacewith(\if(0<=idx & idx<(to-from)+1)
      \then(beta::seqGet(seq, idx + from))
       \else( beta::seqGet(seqEmpty,0)))

      \heuristics(simplify)
  };


  getOfSeqReverse {
      \schemaVar \term Seq seq;
      \schemaVar \term int idx;

\find(beta::seqGet(seqReverse(seq), idx))

\replacewith(beta::seqGet(seq,seqLen(seq)-1-idx))

\heuristics(simplify_enlarging)
  };


  lenNonNegative {
      \schemaVar \term Seq seq;
```

```
    \find(seqLen(seq))

    \add(0 <= seqLen(seq) ==>)

    \heuristics(inReachableStateImplication)
};


lenOfSeqEmpty {
    \find(seqLen(seqEmpty))

    \replacewith(0)

    \heuristics(concrete)
};


lenOfSeqSingleton {
    \schemaVar \term alpha x;

    \find(seqLen(seqSingleton(x)))

    \replacewith(1)

    \heuristics(concrete)
};


lenOfSeqConcat {
    \schemaVar \term Seq seq, seq2;

    \find(seqLen(seqConcat(seq, seq2)))

    \replacewith(seqLen(seq) + seqLen(seq2))

    \heuristics(simplify)
};
```

```
    lenOfSeqSub {
        \schemaVar \term Seq seq;
        \schemaVar \term int from, to;

\find(seqLen(seqSub(seq, from, to)))

\replacewith(
      \if(from <= to)\then((to - from) + 1)\else(0))

 \heuristics(simplify_enlarging)
   };

   lenOfSeqReverse {
        \schemaVar \term Seq seq;

        \find(seqLen(seqReverse(seq)))

        \replacewith(seqLen(seq))

        \heuristics(simplify)
   };

    equalityToSeqGetAndSeqLen {
        \schemaVar \term Seq s, s2;
        \schemaVar \variables int iv;

        \find(s = s2)
        \varcond(\notFreeIn(iv, s, s2))

\replacewith(seqLen(s) = seqLen(s2)
   & \forall iv; (0 <= iv & iv < seqLen(s)
                   ->
        any::seqGet(s, iv) = any::seqGet(s2, iv)))
   };


   equalityToSeqGetAndSeqLenLeft {
        \schemaVar \term Seq s, s2;
```

```
        \schemaVar \variables int iv;

        \find(s = s2 ==>)
        \varcond(\notFreeIn(iv, s, s2))

\add(seqLen(s) = seqLen(s2)
   & \forall iv; (0 <= iv & iv < seqLen(s)
         ->
     any::seqGet(s, iv) = any::seqGet(s2, iv)) ==>)

   \heuristics(inReachableStateImplication)
    };

    equalityToSeqGetAndSeqLenRight {
        \schemaVar \term Seq s, s2;
        \schemaVar \variables int iv;

        \find(==> s = s2)
        \varcond(\notFreeIn(iv, s, s2))

\replacewith(==> seqLen(s) = seqLen(s2)
  & \forall iv; (0 <= iv & iv < seqLen(s)
       ->
     any::seqGet(s, iv) = any::seqGet(s2, iv)))

   \heuristics(simplify_enlarging)
    };

    //----------------------------------------
    //EQ versions of axioms
    // (these are lemmata)
    //----------------------------------------

    getOfSeqSingletonEQ {
        \schemaVar \term alpha x;
        \schemaVar \term int idx;
        \schemaVar \term Seq EQ;
```

```
        \assumes(seqSingleton(x) = EQ ==>)
        \find(beta::seqGet(EQ, idx))
        \sameUpdateLevel

        \replacewith(\if(idx = 0)
                     \then((beta)x)
                     \else(beta::seqGet(seqEmpty, 0)))

        \heuristics(simplify)
    };

     getOfSeqConcatEQ {
        \schemaVar \term Seq seq, seq2;
        \schemaVar \term int idx;
        \schemaVar \term Seq EQ;

        \assumes(seqConcat(seq, seq2) = EQ ==>)
        \find(beta::seqGet(EQ, idx))
        \sameUpdateLevel

\replacewith(
  \if(idx < seqLen(seq))
  \then(beta::seqGet(seq, idx))
  \else(beta::seqGet(seq2, idx - seqLen(seq))))

        \heuristics(simplify_enlarging)
    };

    getOfSeqSubEQ {
        \schemaVar \term Seq seq;
        \schemaVar \term int idx, from, to;
        \schemaVar \term Seq EQ;

        \assumes(seqSub(seq, from, to) = EQ ==>)
        \find(beta::seqGet(EQ, idx))
        \sameUpdateLevel

        \replacewith(beta::seqGet(seq, idx + from))
```

```
        \heuristics(simplify)
    };

     getOfSeqReverseEQ {
        \schemaVar \term Seq seq;
        \schemaVar \term int idx;
        \schemaVar \term Seq EQ;

        \assumes(seqReverse(seq) = EQ ==>)
        \find(beta::seqGet(EQ, idx))
        \sameUpdateLevel

\replacewith(beta::seqGet(seq, seqLen(seq) - 1 - idx))

\heuristics(simplify_enlarging)
    };

    lenOfSeqEmptyEQ {
        \schemaVar \term alpha x;
        \schemaVar \term Seq EQ;

        \assumes(seqEmpty = EQ ==>)
        \find(seqLen(EQ))
        \sameUpdateLevel

        \replacewith(0)

        \heuristics(concrete)
    };

    lenOfSeqSingletonEQ {
        \schemaVar \term alpha x;
        \schemaVar \term Seq EQ;

        \assumes(seqSingleton(x) = EQ ==>)
        \find(seqLen(EQ))
        \sameUpdateLevel
```

```
        \replacewith(1)

        \heuristics(concrete)
    };

    lenOfSeqConcatEQ {
        \schemaVar \term Seq seq, seq2;
        \schemaVar \term Seq EQ;

        \assumes(seqConcat(seq, seq2) = EQ ==>)
        \find(seqLen(EQ))
        \sameUpdateLevel

        \replacewith(seqLen(seq) + seqLen(seq2))

        \heuristics(simplify)
    };

    lenOfSeqSubEQ {
        \schemaVar \term Seq seq;
        \schemaVar \term int from, to;
        \schemaVar \term Seq EQ;

        \assumes(seqSub(seq, from, to) = EQ ==>)
        \find(seqLen(EQ))
        \sameUpdateLevel

\replacewith(
  \if(from <= to)\then((to - from) + 1)\else(0))

\heuristics(simplify_enlarging)
    };


    lenOfSeqReverseEQ {
        \schemaVar \term Seq seq;
        \schemaVar \term Seq EQ;
```

```
    \assumes(seqReverse(seq) = EQ ==>)
    \find(seqLen(EQ))
    \sameUpdateLevel

    \replacewith(seqLen(seq))

    \heuristics(simplify)
};

//------------------------------------------------------------
//lemmata for seqEmpty
//------------------------------------------------------------

concatWithEmpty1 {
    \schemaVar \term Seq seq;

    \find(seqConcat(seq, seqEmpty))

    \replacewith(seq)

    \heuristics(concrete)
};

concatWithEmpty2 {
    \schemaVar \term Seq seq;

    \find(seqConcat(seqEmpty, seq))

    \replacewith(seq)

    \heuristics(concrete)
};

//------------------------------------------------------------
//lemma for casts
//------------------------------------------------------------

castedGetAny {
    \schemaVar \term Seq seq;
```

```
        \schemaVar \term int idx;

        \find((beta)any::seqGet(seq, idx))

        \replacewith(beta::seqGet(seq, idx))

        \heuristics(simplify)
    };
}
```

4.3 Taclets for bSum

```
\schemaVariables{
 \term int subsumLeft, subsumRightBigger,
           subsumRightSmaller, subsumCoeffBigger,
           subsumCoeffSmaller;
}

\rules {
//----------------------------------------
// rules for bounded sums
//----------------------------------------

 bsum_split {
        \find(bsum{uSub;} (i0, i2, t))
        \varcond ( \notFreeIn(uSub1, i0),
                   \notFreeIn(uSub1, i1),
                   \notFreeIn(uSub1, i2),
                   \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i1),
                   \notFreeIn(uSub, i2),
                   \notFreeIn(uSub1, t) )
  \replacewith(\if(i0<=i1 & i1<=i2)
    \then(bsum{uSub;}(i0, i1, t) +
       bsum{uSub1;}(i1,i2,{\subst uSub; uSub1}t))
    \else(bsum{uSub;}(i0, i2, t)))
 };
```

```
 bsum_commutative_associative {
        \find(bsum{uSub;} (i0, i2, t+t2))
        \varcond ( \notFreeIn(uSub1, i0),
                   \notFreeIn(uSub1, i2),
                   \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2),
                   \notFreeIn(uSub1, t2) )
 \replacewith(bsum{uSub;}(i0, i2, t) +
        bsum{uSub1;}(i0, i2, {\subst uSub; uSub1}t2))
        \heuristics(simplify)
 };

bsum_induction_upper {
        \find(bsum{uSub;} (i0, i2, t))
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2))
  \replacewith(bsum{uSub;} (i0, i2-1, t) +
    \if(i0<i2)
     \then({\subst uSub; (INT2)(i2-1)}t)
     \else(0))
 };

 bsum_induction_upper2 {
        \find(bsum{uSub;} (i0, i2, t))
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2))
  \replacewith(bsum{uSub;} (i0, i2+1, t) -
    \if(i0<i2+1)
     \then({\subst uSub; (INT2)(i2)}t)
     \else(0))
 };

 bsum_induction_upper_concrete {
        \find(bsum{uSub;} (i0, 1+i2, t))
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2))
 \replacewith(bsum{uSub;} (i0, i2, t) +
   \if(i0<=i2)
     \then({\subst uSub; (INT2)(i2)}t)
```

```
      \else(0))
   \heuristics(simplify)
  };

  bsum_induction_upper2_concrete {
         \find(bsum{uSub;} (i0, -1+i2, t))
         \varcond ( \notFreeIn(uSub, i0),
                    \notFreeIn(uSub, i2))
\replacewith(bsum{uSub;} (i0, i2, t) -
     \if(i0<i2)
      \then({\subst uSub; (INT2)(i2-1)}t)
      \else(0))
\heuristics(simplify)
  };

   bsum_induction_lower {
         \find(bsum{uSub;} (i0, i2, t))
         \varcond ( \notFreeIn(uSub, i0),
                    \notFreeIn(uSub, i2))
   \replacewith(bsum{uSub;} (i0+1, i2, t) +
     \if(i0<i2)
      \then({\subst uSub; (INT2)(i0)}t)
      \else(0))
  };

   bsum_induction_lower_concrete {
         \find(bsum{uSub;} (-1+i0, i2, t))
         \varcond ( \notFreeIn(uSub, i0),
                    \notFreeIn(uSub, i2))
 \replacewith(bsum{uSub;} (i0, i2, t) +
     \if(-1+i0<i2)
      \then({\subst uSub; (INT2)(-1+i0)}t)
       \else(0))
   \heuristics(simplify)
  };

  bsum_induction_lower2 {
         \find(bsum{uSub;} (i0, i2, t))
         \varcond ( \notFreeIn(uSub, i0),
```

```
                    \notFreeIn(uSub, i2))
 \replacewith(bsum{uSub;} (i0-1, i2, t) -
    \if(i0-1<i2)
     \then({\subst uSub; (INT2)(i0-1)}t)
     \else(0))
 };

 bsum_induction_lower2_concrete {
        \find(bsum{uSub;} (1+i0, i2, t))
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2))
  \replacewith(bsum{uSub;} (i0, i2, t) -
    \if(i0<i2)
     \then({\subst uSub; (INT2)(i0)}t)
     \else(0))
  \heuristics(simplify)
 };

  bsum_zero_right {
        \find(==> bsum{uSub;} (i0, i2, t)=0)
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2))
 \add(==> \forall uSub;
  {\subst uSub; uSub}(uSub>=i0 & uSub<i2 -> t=0))
\heuristics(comprehensions)
 };

 bsum_distributive {
        \find(bsum{uSub;} (i0, i2, t*t1))
        \varcond ( \notFreeIn(uSub, i0),
                   \notFreeIn(uSub, i2),
                   \notFreeIn(uSub, t1))
        \replacewith(bsum{uSub;} (i0, i2, t)*t1)
 };

 bsum_equal_split1 {
        \find(==> bsum{uSub1;} (i0, i1, t1)
                = bsum{uSub2;} (i0, i2, t2))
        \varcond (\notFreeIn(uSub1, i0),
```

```
                    \notFreeIn(uSub1, i1),
                    \notFreeIn(uSub1, i2),
                    \notFreeIn(uSub1, t2),
                    \notFreeIn(uSub2, i2),
                    \notFreeIn(uSub2, i1),
                    \notFreeIn(uSub2, t1),
                    \notFreeIn(uSub2, i0))
\add(==> i0<=i1 & i0<=i2 &
 \if(i1<i2)
  \then(bsum{uSub1;} (i0, i1, t1-
    {\subst uSub2; uSub1}t2) =
      bsum{uSub2;}(i1,i2,t2))
  \else(bsum{uSub1;} (i2, i1, t1) =
    bsum{uSub2;}(i0,i2,t2-{\subst uSub1; uSub2}t1)))
 \heuristics(comprehensions)
 };

 bsum_equal_split2 {
        \assumes(bsum{uSub1;} (i0, i1, t1) = i ==>)
        \find(==>  bsum{uSub2;} (i0, i2, t2) = i)
        \varcond (\notFreeIn(uSub1, i0),
                    \notFreeIn(uSub1, i1),
                    \notFreeIn(uSub1, i2),
                    \notFreeIn(uSub1, t2),
                    \notFreeIn(uSub2, i2),
                    \notFreeIn(uSub2, t1),
                    \notFreeIn(uSub2, i1),
                    \notFreeIn(uSub2, i0))
\add(==> i0<=i1 & i0<=i2 &
 \if(i2<i1)
  \then(bsum{uSub1;} (i2, i1, t1) =
    bsum{uSub2;} (i0, i2, t2-{\subst uSub1; uSub2}t1))
  \else(bsum{uSub1;}(i0,i1,t1-{\subst uSub2; uSub1}t2)
        = bsum{uSub2;} (i1, i2, t2)))
 \heuristics(comprehensions)
 };

 bsum_equal_split3 {
```

```
        \find(==> bsum{uSub1;} (i1, i0, t1) =
                  bsum{uSub2;} (i2, i0, t2))
        \varcond (\notFreeIn(uSub1, i0),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub1, i2),
                  \notFreeIn(uSub1, t2),
                  \notFreeIn(uSub2, i2),
                  \notFreeIn(uSub2, i1),
                  \notFreeIn(uSub2, t1),
                  \notFreeIn(uSub2, i0))
\add(==> i1<=i0 & i2<=i0 &
 \if(i1<i2)
  \then(bsum{uSub1;} (i1, i2, t1) =
     bsum{uSub2;} (i2, i0, t2-{\subst uSub1; uSub2}t1))
  \else(bsum{uSub1;} (i1,i0,t1-{\subst uSub2; uSub1}t2)
        = bsum{uSub2;} (i2, i1, t2)))
\heuristics(comprehensions)
 };

 bsum_equal_split4 {
        \assumes(bsum{uSub1;} (i1, i0, t1) = i ==>)
        \find(==>  bsum{uSub2;} (i2, i0, t2) = i)
        \varcond (\notFreeIn(uSub1, i0),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub1, i2),
                  \notFreeIn(uSub1, t2),
                  \notFreeIn(uSub2, i2),
                  \notFreeIn(uSub2, t1),
                  \notFreeIn(uSub2, i1),
                  \notFreeIn(uSub2, i0))
 \add(==> i1<=i0 & i2<=i0 &
 \if(i2<i1)
  \then(bsum{uSub1;}(i1,i0,t1-{\subst uSub2; uSub1}t2)
        = bsum{uSub2;} (i2, i1, t2))
  \else(bsum{uSub1;} (i1, i2, t1) =
     bsum{uSub2;}(i2,i0,t2-{\subst uSub1;uSub2}t1)))
  \heuristics(comprehensions)
 };
```

```
bsum_split_in_three {
        \find(bsum{uSub;} (i0, i2, t))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i1),
                  \notFreeIn(uSub1, t),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub, i0),
                  \notFreeIn(uSub1, i2))
"Precondition": \add(==> (i0<=i1 & i1<i2));
"Splitted Sum": \replacewith(
        bsum{uSub;} (i0, i1, t) +
        {\subst uSub; (INT2) i1} t +
        bsum{uSub1;} (i1+1, i2, {\subst uSub; uSub1}t))
};

bsum_empty {
        \find(bsum{uSub;} (i0, i1, t))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i0),
                  \notFreeIn(uSub, i1))
        "Precondition": \add(==> i1<=i0);
        "Empty Sum": \replacewith(0)
};

bsum_one_summand {
        \find(bsum{uSub;} (i0, i1, t))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i0),
                  \notFreeIn(uSub, i1))
\replacewith(\if(i0+1=i1)
               \then({\subst uSub; (INT2) i0} t)
               \else(bsum{uSub;} (i0, i1, t)))
};

bsum_empty_concrete1 {
        \find(==>bsum{uSub;} ( Z(iz),  Z(jz), t)=0)
        \varcond (\notFreeIn(uSub, iz),
                  \notFreeIn(uSub, jz))
        \add(==> Z(jz)<=Z(iz))
        \heuristics(simplify)
};
```

```
bsum_empty_concrete2 {
        \find(bsum{uSub;} ( Z(iz), Z(neglit(jz)), t))
        \varcond (\notFreeIn(uSub, iz),
                  \notFreeIn(uSub, jz))
 \replacewith(
    \if(Z(neglit(jz))<=Z(iz))
    \then(0)
    \else(bsum{uSub;} ( Z(iz), Z(neglit(jz)), t)))
 \heuristics(simplify)
};

bsum_zero {
        \find(bsum{uSub;} (i0, i1, 0))
        \varcond (\notFreeIn(uSub, i0),
                  \notFreeIn(uSub, i1))
        \replacewith(0)
        \heuristics(simplify)
};

bsum_lower_equals_upper {
        \find(bsum{uSub;} (i0, i0, t))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i0))
        \replacewith(0)
        \heuristics(simplify)
};

//this case occurs when translating \num_of
bsum_positive1 {
        \find(bsum{uSub;} (i0, i1,
            \if(b)\then(1)\else(0)))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i0),
                  \notFreeIn(uSub, i1))
\add(bsum{uSub;}(i0,i1,\if(b)\then(1)\else(0))>=0 ==>)
\heuristics(simplify)
};

//this case occurs when translating \num_of
bsum_positive2 {
        \find(bsum{uSub;} (i0, i1,
```

```
            \if(b)\then(0)\else(1)))\sameUpdateLevel
        \varcond (\notFreeIn(uSub, i0),
                  \notFreeIn(uSub, i1))
\add(bsum{uSub;}(i0,i1,\if(b)\then(0)\else(1))>=0 ==>)
\heuristics(simplify)
};

equal_bsum1 {
        \find(==> bsum{uSub1;} (i0, i1, t1) =
                  bsum{uSub2;} (i0, i1, t2))
        \varcond (\notFreeIn(uSub2, t1),
                  \notFreeIn(uSub1, t2),
                  \notFreeIn(uSub1, i0),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub2, i0),
                  \notFreeIn(uSub2, i1))
\add(==>\forall uSub1; ((uSub1>=i0 & uSub1<i1) ->
                         t1=({\subst uSub2; uSub1}t2)))
\heuristics(comprehensions_high_costs)
};

equal_bsum2 {
        \assumes(bsum{uSub1;} (i0, i1, t1) = i ==>)
        \find(==>  bsum{uSub2;} (i0, i1, t2) = i)
        \varcond (\notFreeIn(uSub2, t1),
                  \notFreeIn(uSub1, t2),
                  \notFreeIn(uSub1, i0),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub2, i0),
                  \notFreeIn(uSub2, i1))
 \add(==>\forall uSub1; ((uSub1>=i0 & uSub1<i1) ->
                         t1=({\subst uSub2; uSub1}t2)))
        \heuristics(comprehensions_high_costs)
};

equal_bsum3 {
        \assumes(bsum{uSub1;} (i0, i1, t1) = i,
                 bsum{uSub2;} (i0, i1, t2) = j ==>)
        \find(==>  j = i)
```

```
        \varcond (\notFreeIn(uSub2, t1),
                  \notFreeIn(uSub1, t2),
                  \notFreeIn(uSub1, i0),
                  \notFreeIn(uSub1, i1),
                  \notFreeIn(uSub2, i0),
                  \notFreeIn(uSub2, i1))
 \add(==>\forall uSub1; ((uSub1>=i0 & uSub1<i1) ->
                        t1=({\subst uSub2; uSub1}t2)))
        \heuristics(comprehensions_high_costs)
 };

 equal_bsum_zero_cut {
        \find(==>  bsum{uSub1;} (i0, i1, t1) =
                   bsum{uSub2;} (i2, i3, t2)*t)
        \add( ==> bsum{uSub1;} (i0, i1, t1)=0);
        \add( bsum{uSub1;} (i0, i1, t1)=0 ==>)
        \heuristics(comprehensions_high_costs)
 };

 pullOutbsum1 {
        \find(bsum{uSub1;} (i0, i1, t1) >= t ==>)
           \varcond ( \new(sk, \dependingOn(t1)),2
                      \new(sk, \dependingOn(i0)),
                      \new(sk, \dependingOn(i1)) )
        \replacewith (sk >= t ==>)
        \add ( bsum{uSub1;} (i0, i1, t1) = sk ==>)
        \heuristics(simplify)
 };

 pullOutbsum2 {
        \find(bsum{uSub1;} (i0, i1, t1) <= t ==>)
           \varcond ( \new(sk, \dependingOn(t1)),
                      \new(sk, \dependingOn(i0)),
                      \new(sk, \dependingOn(i1)) )
        \replacewith (sk <= t ==>)
        \add ( bsum{uSub1;} (i0, i1, t1) = sk ==>)
        \heuristics(simplify)
 };
}
```

Bibliography

[1] E. Aaron and D. Gries. Formal justification of underspecification for S5. *Inf. Process. Lett.*, 64(3):115–121, 1997.

[2] A. Armando, P. Baumgartner, and G. Dowek, editors. *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*. Springer, 2008.

[3] C. Barrett, S. Berezin, I. Shikanian, M. Chechik, A. Gurfinkel, and D. L. Dill. A practical approach to partial functions in CVC Lite. In *PDPAR'04 Workshop, Cork, Ireland*, volume 125 of *Electronic Notes in Computer Science*, pages 13–23, July 2005.

[4] B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach.* LNCS 4334. Springer-Verlag, 2007.

[5] P. Behm, L. Burdy, and J.-M. Meynadier. Well defined B. In D. Bert, editor, *B*, volume 1393 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 1998.

[6] D. Bruns. Formal semantics for the Java Modeling Language. Diploma thesis, Universität Karlsruhe, June 2009.

[7] Á. Darvas, F. Mehta, and A. Rudich. Efficient well-definedness checking. In Armando et al. [2], pages 100–115.

[8] D. Gries and F. B. Schneider. Avoiding the undefined by underspecification. In van Leeuwen [19], pages 366–373.

[9] R. Hähnle. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL*, 13(4):415–433, 2005.

[10] V. Klasen. Verifying Dijstra's algorithm with KeY. Diplomarbeit, Universität Koblenz, March 2010.

[11] M. Krötzsch. A counterexample on conservative extensions. private communication, 2010.

[12] G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Müller, J. Kiniry, and P. Chalin. *JML Reference Manual*, september 2009.

[13] G. T. Leavens and J. M. Wing. Protection from the underpsecified. Technical Report TR# 96-04, Iowa State University, April 1996.

[14] G. T. Leavens and J. M. Wing. Protective interface specifications. *Formal Asp. Comput.*, 10(1):59–75, 1998.

[15] K. R. M. Leino. Specification and verification of object-oriented software. Markoberdorf International Summer School, Lecture Notes, 2008.

[16] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010.

[17] A. Rudich, Á. Darvas, and P. Müller. Checking well-formedness of pure-method specifications. In J. Cuéllar, T. S. E. Maibaum, and K. Sere, editors, *FM*, volume 5014 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2008.

[18] P. H. Schmitt. Formal specification and verification. Lecture Notes, 2011.

[19] J. van Leeuwen, editor. *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*. Springer, 1995.