A Computer-Assisted Proof of the Bellman-Ford Lemma

Peter H. Schmitt

KIT, Karlsruhe Institute of Technology

March 31, 2011
Please note:
This Report has been published on the Internet under the following Creative Commons License:
http://creativecommons.org/licenses/by-nc-nd/3.0/de.
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

1 A Computer-Assisted Proof 2
  1.1 Basic Definitions ........................................ 2
  1.2 The Bellman-Ford Lemma .................................. 6
  1.3 Auxiliary Concept ........................................ 8
  1.4 Proof .................................................... 8
    1.4.1 Proof of Claim 1 ..................................... 9
    1.4.2 Proof of Claim 2 ..................................... 13
  1.5 Conclusion and Outlook .................................. 15

2 Partial Functions 19
  2.1 Motivation and Introduction .............................. 19
  2.2 A Theory of Underspecification ........................... 22
  2.3 Total Formulas ............................................ 27
  2.4 The Transformation \( \mathcal{Y} \) ............................ 36
  2.5 Examples ................................................ 43
  2.6 A Theory of Sequences .................................... 51
  2.7 An Overspecified Theory of Sequences .................... 62
  2.8 Closing Remarks .......................................... 67

3 Conservative Extension 69
  3.1 Review of Basic Definitions ............................. 69
  3.2 Digression ............................................... 74

4 Taclets 78
  4.1 Taclets for some Axioms ................................. 78
  4.2 Taclets for the Sequence Data Type ..................... 79
  4.3 Taclets for bSum ......................................... 89
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

\[ \forall \text{node } n; (\neg \text{edge}(n, n)) \] (1.1)

\[ \forall \text{node } m; (\forall \text{node } n; (\text{edge}(n, m) \rightarrow w(n, m) > 0)) \] (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 \( \text{node} \). The interpretation of the sort \( \text{int} \) 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 \( \text{edge} \).

3. \( w : E \rightarrow \mathbb{Z} \) a function that associates an integer with every edge.
   Here \( \mathbb{Z} \) is the interpretation of sort \( \text{int} \).

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 \( G = (V, E, w) \) 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 \( G \), 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 \( \text{fwpath}(Seq) \) and the defining axiom
∀Seq seq; (fwpath(seq) < − >
∀int iv; ((0 <= iv & iv < seqLen(seq) − 1)− >
    edge(node :: seqGet(seq, iv), node :: seqGet(seq, iv + 1)) &
    node :: instance(any : seqGet(seq, iv)) &
    node :: instance(any : seqGet(seq, seqLen(seq) − 1))) (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 →-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

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

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 ∀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

\[
\frac{b(\langle \rangle) \quad \forall obj x; (b(s) \rightarrow b(s + \langle x\rangle))}{\forall seq s; b(s)}
\]
Here, $s + \langle x \rangle$ is shorthand for $\text{seqConcat}(s, \text{seqSingleton}(x))$.

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

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

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

We have presented the induction rules in mathematical style, the taclet formalisations may be found in Subsection 4.1. Most of the time when proving a formula $\forall \text{seq } s; b(s)$ the induction rule is used to first prove $\forall \text{seq } s; a(s)$ and then to show that $\forall \text{seq } s; a(s) \rightarrow \forall \text{seq } s; b(s)$ 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 $\text{pw}(s)$, is the sum of the weights of its edges, i.e., $\text{pw}(s) = \Sigma_{i=0}^{k-1} w((v_i, v_{i+1}))$.

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 $\text{pw}(s) \leq \text{pw}(t)$. 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 $\text{bsum}$ available with the KeY system. The syntax is

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

where

1. $iv$ is a variable of type $\text{int}$
2. $t_1, t_2$ are terms of type $\text{int}$ that do not contain $iv$
3. $t$ is a term of type $\text{int}$ that typically will, but need not, contain the variable $iv$. 
The term $bsum\{iv\}(t_1, t_2, t)$ is evaluated in a structure $A$ and variable assignment $\beta$ as

$$bsum\{iv\}(t_1, t_2, t)(A,\beta) = \begin{cases} 0 & \text{if } b \leq a \\ \sum_{i=b}^{a-1} c(i) & \text{otherwise} \end{cases}$$

with

1. $a = t_1(A,\beta)$
2. $b = t_2(A,\beta)$
3. $c(i) = t(A,\beta_i)$ 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:

$$\forall Seq\, seq; \quad \text{pw}(seq) = bsum\{iv\}(0, \text{seqLen}(seq) - 1, w(\text{node :: seqGet(seq,iv)}, \text{node :: seqGet(seq,iv+1)}))) \quad (1.4)$$

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

Definition 1.4 entails the following special cases:

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

### 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. \( d(\text{start}) = 0 \)

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

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 \text{start} to \( n \) in \( G \).

The properties 2 and 3 are called the Bellman-Ford Equations. They are sometimes summarized as

\[ \forall m \in V \setminus \{ \text{start} \}(d(m) = \min \{ d(n) + w(n, m) \mid (n, m) \in E \}) \]

The assumptions of the lemma are formalized as follows.

\[ \forall \text{node } n; (d(n) \geq 0) \]  

(1.5)

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

\[ d(\text{start}) = 0 \]  

(1.6)

\[ \forall \text{node } m; (m! = \text{start} \rightarrow \exists \text{node } n; (\text{edge}(n, m) \land d(m) = d(n) + w(n, m))) \]  

(1.7)

First Bellman-Ford equation.

\[ \forall \text{node } m; (\forall \text{node } n; (\text{edge}(n, m) \rightarrow d(m) \leq d(n) + w(n, m))) \]  

(1.8)

Second Bellman-Ford equation.

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

\[ (\forall \text{node } m; (m! = \text{start} \rightarrow (\exists \text{Seq } s; (\text{fwpath}(s) \land \text{node :: seqGet}(s, 0) = \text{start} \land \text{node :: seqGet}(s, \text{seqLen}(s) - 1) = m \land \text{pw}(s) = d(m)))) \)  

(1.9)

\[ \forall \text{node } m; (\forall \text{Seq } s; ((\text{fwpath}(s) \land \text{node :: seqGet}(s, 0) = \text{start} \land \text{node :: seqGet}(s, \text{seqLen}(s) - 1) = m \land d(m) \leq \text{pw}(s))) \)  

(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 \text{start}. This restriction could be waived by considering \( d : \text{Node} \rightarrow \mathbb{N} \cup \{ \infty \} \). 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:

\[
\forall \text{Seqs}; (dpath(s) \leftrightarrow \text{fwpath}(s) \& \forall \text{int } i; (0 \leq i < \text{seqLen}(s) - 1 \rightarrow (d(s[i+1]) = d(s[i]) + w(s[i], s[i+1]))) (1.11)
\]

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:

\[(1.1) \& (1.2) \& (1.3) \& (1.4) \& (1.5) \& (1.6) \& (1.7) \& (1.8) \& (1.11) \rightarrow (1.9)
\]

\[(1.1) \& (1.2) \& (1.3) \& (1.4) \& (1.5) \& (1.6) \& (1.7) \& (1.8) \& (1.11) \rightarrow (1.10)
\]

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

\[(1.1) \& (1.2) \& (1.5) \& (1.6) \& (1.7) \& (1.8) \implies (1.9)
\]

\[(1.1) \& (1.2) \& (1.5) \& (1.6) \& (1.7) \& (1.8) \implies (1.10)
\]

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.

<table>
<thead>
<tr>
<th>Claim</th>
<th>input file name</th>
<th>proof file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.12)</td>
<td>Lemma1.key</td>
<td>Lemma1.key.proof</td>
</tr>
<tr>
<td>(1.13)</td>
<td>Lemma2.key</td>
<td>Lemma2.key.proof</td>
</tr>
<tr>
<td>(1.14)</td>
<td>Lemma3.key</td>
<td>Lemma3.key.proof</td>
</tr>
<tr>
<td>(1.15)</td>
<td>Lemma4.key</td>
<td>Lemma4.key.proof</td>
</tr>
<tr>
<td>(1.16)</td>
<td>Lemma5.key</td>
<td>Lemma5.key.proof</td>
</tr>
<tr>
<td>(1.10)</td>
<td>seqBFTheoremPart2.key</td>
<td>seqBFTheoremPart2.key.proof</td>
</tr>
</tbody>
</table>

### 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.

\[
\forall Seq s; (fdpath(s) \rightarrow d(node :: seqGet(s, seqLen(s) - 1)) \geq seqLen(s) - 1 | seqLen(s) = 0)
\]

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 \(d(s_{n-1}) = d(s_0) + \sum_{i=0}^{n-2} w(s_i, s_{i+1})\). 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

$$(1.2) \& (1.3) \& (1.5) \& (1.11) \rightarrow (1.12)$$

is true.

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 $\forall m ; (\exists s ; (\ldots))$ is proved by instantiating $s$ by $seqSingleton(m_0)$ where $m_0$ is the Skolem constant for $\forall m$. 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 $\forall m ; \exists s ; (\phi(m, s, i) \Rightarrow \forall m ; \exists s ; (\phi(m, s, i + 1))$. Let $m_1$ be the Skolem constant for the righthand side $\forall m$. Instantiate the lefthand side $\forall m$ 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 distinction whether $node :: seqGet(s_0, 0) = start$ or not. If $node :: seqGet(s_0, 0) = start$ then we can instantiate the quantifier $\exists s ; (\phi(m_1, s, i + 1))$ with $s_0$. The
sequence already starts with the \textit{start} node and need not be extended. After manually unfolding all \textit{fdpath} and \textit{fwpath} functions \textit{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 \texttt{n}_0 with

\begin{align*}
\text{edge}(n_0, \text{node} :: \text{seqGet}(s_0,0)) \text{ and } \\
d(\text{node} :: \text{seqGet}(s_0,0)) = d(n_0) + w(n_0, \text{node} :: \text{seqGet}(s_0,0))
\end{align*}

For this node :: seqGet(s_0,0) ! = start is necessary. Now we are able to instantiate the existential quantifier \( \exists \text{seq} s; (\phi(m_1, s, i+1)) \) on the righthand side by seqConcat(seqSingleton(n_0), s_0) and after the usual unfolding of \textit{fdpath} and \textit{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{align*}
\forall \text{node} \ m; (\exists \text{Seq} \ s; ( & \\
\text{fdpath}(s) \ & \\
\text{seqLen}(s) \geq 1 \ & \\
\text{node} \ :: \text{seqGet}(s, \text{seqLen}(s) - 1) = m \ & \\
\text{node} \ :: \text{seqGet}(s, 0) = \text{start})) \tag{1.14}
\end{align*}

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 \texttt{m}_0 be the Skolem constant introduced by eliminating the universal quantifier \( \forall \text{node} \ m; (\ldots) \) on the righthand side and instantiating \( \forall \text{int} \ i; (\ldots) \) on the lefthand side with \( d(\texttt{m}_0) + 2 \) then Z3 can solve the problem immediately (the time shows 0.0sec). \textit{KeY} on the other hand cannot do this. Only after unfolding all existential quantifiers and instantiating all universal quantifiers, about a handful of interactions, \textit{KeY} finds the rest of the proof automatically.

\begin{align*}
\forall \text{Seq} \ s; ( & \\
(\text{fdpath}(s) \ & \text{seqLen}(s) \geq 2 \ & \text{node} :: \text{seqGet}(s, 0) = \text{start}) \ & \\
\rightarrow & \\
d(\text{node} :: \text{seqGet}(s, \text{seqLen}(s) - 1)) = pw(s)) \tag{1.15}
\end{align*}

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:

\[
\forall \text{int } i; \forall \text{Seq } s; (\text{fdpath}(s) \& \text{seqLent}(s) = i \& \text{seqLen}(s) \geq 1 \& \text{node :: seqGet}(s, 0) = \text{start} \\
\quad \quad - > d(\text{node :: seqGet}(s, \text{sub(seqLen}(s), 1))))
= \text{pw}(s))
\]

The initial case is trivial since \(\text{seqLen}(s) = 0\&\text{seqLen}(s) \geq 1\) on the left-hand side of the implication to be proved is contradictory. KeY finds this automatically. After interactive instantiation of \(i = \text{seqLen}(s_0)\) where \(s_0\) is the Skolem constant for the quantifier \(\forall \text{Seq } s\) 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 \(i_0 \geq 0\). At this point it makes sense to distinguish 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 \(\text{seqSub}(s_4, 0, i_0 - 1)\). Before the crucial part of the induction hypothesis can be used, we have to show that

1. \(\text{seqSub}(s_4, 0, i_0)\) is an \(d\)-path given that \(s_4\) is a \(d\)-path
2. \(\text{seqLen}(\text{seqSub}(s_4, 0, i_0)) = i_0\)
3. \(\text{seqLen}(\text{seqSub}(s_4, 0, i_0)) \geq 1\)
4. \(\text{node :: seqGet}(\text{seqSub}(s_4, 0, i_0), 0) = \text{start}\) given that \(\text{node :: seqGet}(s_4, 0) = \text{start}\).

It only takes unfolding of the definitions of \(\text{pdpath}\) and \(\text{fpath}\) 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

\[
d(\text{node :: seqGet}(s_4, i_0)) = d(\text{node :: seqGet}(s_4, i_0 - 1)) + w(\text{node :: seqGet}(s_4, i_0 - 1), \text{node :: seqGet}(s_4, i_0))
\]

plus

\[
d(\text{node :: seqGet}(s_4, i_0 - 1)) = \text{pw}(\text{seqSub}(s_4, 0, i_0 - 1))
\]
and need to prove
\[ d(\text{node} :: \text{seqGet}(s_4, i_0)) = \text{pw}(s_4) \]

After unfolding \( \text{pw} \) the proof completes almost automatically. Surprisingly the taclet \texttt{bsum_one_summand} has to be triggered interactively.
Statistics 715 nodes, 9 branches, 74 interactions.

Finally we show
\[ (1.14) & (1.15) \Rightarrow (1.9) \] (1.16)

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>nodes</th>
<th>branches</th>
<th>interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.2) &amp; (1.3) &amp; (1.5) &amp; (1.11) ( \Rightarrow ) (1.12)</td>
<td>4227</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>(1.3) &amp; (1.11) &amp; (1.7) ( \Rightarrow ) (1.13)</td>
<td>610</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>(1.2) &amp; (1.12) &amp; (1.13) ( \Rightarrow ) (1.14)</td>
<td>83</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>(1.3) &amp; (1.4) &amp; (1.6) &amp; (1.11) ( \Rightarrow ) (1.15)</td>
<td>715</td>
<td>9</td>
<td>74</td>
</tr>
<tr>
<td>(1.14) &amp; (1.15) ( \Rightarrow ) (1.9)</td>
<td>82</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Total</td>
<td>5697</td>
<td>36</td>
<td>168</td>
</tr>
</tbody>
</table>

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

\textbf{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 \texttt{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 \( s_0 = \text{seqEmpty} \) and \( s_0 \neq \text{seqEmpty} \).

If \( s_0 = \text{seqEmpty} \) then the claim to be proved reduces to \( d(n_0) \leq \text{pw}(\text{seqSingleton}(n_0)) \). By definition of \( \text{bsum} \) with equal lower and upper bound we get \( \text{pw}(\text{seqSingleton}(n_0)) = 0 \). It takes to interactions to trigger direct cuts to obtain \( n_0 = \text{start} \). KeY finds the assumption \( d(\text{start}) = 0 \) and finishes this case automatically. Note, the induction hypothesis is not used here.

From now on we have \( s_0 \neq \text{seqEmpty} \) 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

\[
d(n_0) \leq \text{pw}(\text{seqConcat}(s_0, \text{seqSingleton}(n_0))).
\]

To prepare the application of the induction hypothesis we unfold the definition of \( \text{pw} \) and split the resulting sum with the taclet \( \text{bsum\_split} \):

\[
d(n_0) \leq \sum_{j=0}^{\text{seqLen}(s_0)-2} w(\text{node::seqGet(seqConcat(s_0, seqSingleton(n_0)), j)},
\text{node::seqGet(seqConcat(s_0, seqSingleton(n_0)), j + 1)})
+w(\text{node::seqGet(s_0, seqLen(s_0) - 1), n_0})
\]

After proving

1. \( \text{fwpath}(s_0) \)
2. \( \text{node::seqGet}(s_0, 0) = \text{start} \)
3. \( \text{seqLen}(s_0) \geq 1 \)

which is done by applying the rule \( \text{implication\_left} \) unfolding for the two occurrences of \( \text{fwpath} \) its definition and let KeY do the rest automatically, we have the induction hypothesis \( d(\text{node::seqGet}(s_0, \text{seqLen}(s_0) - 1)) \leq \text{pw}(s_0) \) at our disposal.

Now we face a serious problem. We have to show that \( \text{bsum\_1} = \text{bsum\_0} \) for

\[
\text{bsum\_1} = \sum_{j=0}^{\text{seqLen}(s_0)-2} w(\text{node::seqGet(seqConcat(s_0, seqSingleton(n_0)), j)},
\text{node::seqGet(seqConcat(s_0, seqSingleton(n_0)), j + 1)})
\]

14
It is a formidable task for symbolic deduction to show that two sums \( \Sigma_{i=0}^{b} t_i \) and \( \Sigma_{i=0}^{b} s_i \), even with the same bounds, are equal. This would, e.g., be true for any permutation \( s_0, \ldots, s_b \) 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 \( \texttt{equal\_bsum3} \) does this. The \texttt{pull\_out} taclet has to be used twice to prepare for an application of \( \texttt{equal\_bsum3} \).

Summarizing, we have at this point

\[
d(n_{1}) \leq bsum_{1}
\]  

with \( n_{1} = \text{node :: seqGet}(s_{0}, \text{seqLen}(s_{0}) - 1) \).

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

\[
d(n_{0}) \leq d(n_{1}) + w(n_{1}, n_{0})
\]  

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

\[
d(n_{0}) \leq bsum_{1} + w(n_{1}, n_{0})
\]

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 assistant 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. Strengthening 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 \texttt{bsum \_one \_summand}

   \begin{verbatim}
   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) \\
   };
   \end{verbatim}

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 \texttt{apply rules automatically here}. This would be in particular useful for exploring the limits of interaction.
In a lot of proof situations I tried whether 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 $d : V \rightarrow \mathbb{N} \cup \{\infty\}$ with the Bellman-Ford equations accordingly modified:

$$\forall m; (m! = \text{start} \& d(m) \neq \infty \rightarrow$$
$$\exists n; (\text{edge}(n, m) \& d(m) = d(n) + w(n, m)))$$

(1.19)

$$\forall m; (\forall n; (\text{edge}(n, m) \& d(n) \neq \infty) \rightarrow$$
$$d(m) \leq d(n) + w(n, m)))$$

(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 \( \text{fix}_d(x) = \text{reach}_\text{start}(x) \) where \( \text{reach}_\text{start}(\text{node}) \) is a new atomic predicate. With this axiomatisation the Bellman-Ford equations take on the form:

\[
\forall \text{node } m; (m! = \text{start} \land \text{reach}_\text{start}(m) \rightarrow \\
\exists \text{node } n; (\text{edge}(n, m) \land d(m) = d(n) + w(n, m))) \tag{1.21}
\]

\[
\forall \text{node } m; (\forall \text{node } n; (\text{edge}(n, m) \land \text{reach}_\text{start}(n) \rightarrow \\
\quad d(m) \leq d(n) + w(n, m))) \tag{1.22}
\]

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

\[
\forall \text{node } m; (m! = \text{start} \land \text{reach}_\text{start}(m) \\
\quad \rightarrow \quad (\exists \text{Seq } s; (\text{fw}(s) \land \\
\quad \text{node } :: \text{seqGet}(s, 0) = \text{start} \land \\
\quad \text{node } :: \text{seqGet}(s, \text{seqLen}(s) - 1) = m \land \\
\quad \text{pw}(s) = d(m)))) \tag{1.23}
\]

\[
\forall \text{node } m; (\forall \text{Seq } s; (\text{fw}(s) \land \\
\quad \text{node } :: \text{seqGet}(s, 0) = \text{start} \land \\
\quad \text{node } :: \text{seqGet}(s, \text{seqLen}(s) - 1) = m) \\
\quad \rightarrow \quad d(m) \leq \text{pw}(s) \land \text{reach}_\text{start}(m)) \tag{1.24}
\]

From 1.23 and 1.24 it follows in particular that \( \text{reach}_\text{start}(m) \) is true if and only if \( m \) is reachable from \text{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 absolutely 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 $1/0 \div 1/0$ is universally valid, while $1/0 \div 2/0$ is not. It may also not be too hard to accept the statement $\forall \text{seqs} \ \exists \text{seq t}(t \div \text{seqSub}(s, -1, -5))$. But, would you have expected that

$$\exists \text{seq s}(\text{seqLen}(s) = 0 \& \text{int :: seqGet(seqSub(s, 1, 2))} = 5)$$

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 $\phi$ a well-definedness condition $wd_\phi$. The intention is that if $T \vdash wd_\phi$ can be derived then $\phi$ is well-defined in all models of the underlying theory $T$. The proof task $T \vdash \phi$ is then augmented to $T \vdash \phi \land wd_\phi$.

The seemingly innocent question *When is a formula well-defined?* does not have a straightforward 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 first-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* referred 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 $\phi \land \psi$ the situation that $\phi$ is false but $\psi$ may be undefined, would we consider $\phi \land \psi$ to be well-defined? What if $\psi$ is false but $\phi$ 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 $\phi \lor \neg \phi$ is
encountered is it immediately evaluated to true, without decending into the recursive truth definition of $\phi$?

The papers [5, 3] take another approach. They start with a 3-valued logic and call a formula $\phi$ well-defined if it can be proved that $\phi$ 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 $\phi$ is well-defined with respect to a theory $Th$ if the truth value of $\phi$ in any model of $Th$ does not depend on the values of functions $f$ outside their fixed value formula $\text{fix}_f$.

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

What we have discussed so far concerns formulas $\phi$ 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

$$\forall seq s \forall int i, j (i > j \rightarrow \text{seqLen}(\text{seqSub}(s, i, j)) = 0)$$

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

$$\forall < T > \forall seqT \forall int i \forall T v \forall int len\left(0 \leq \text{len} \rightarrow \text{seq#Length} (\text{seq#Build}(s, i, v, \text{len})) = \text{len}\right)$$

The function $\text{seq#Build}(s, i, v, \text{len})$ returns the subsequence of $s$ from index 0 to index $\text{len} - 1$ that is in addition updated at index $i$ by the value
This axiom is also not well-defined since the fixed-value formula for $\text{Seq\#Build}(s, i, v, \text{len})$ is $(0 \leq \text{len} < \text{Seq\#Length}(s)) \land (0 \leq i < \text{len})$.

Including the full domain restrictions would make the axioms considerably more bulky and reasoning 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, $\Sigma_p$, and total symbols $\Sigma_t$.

For every $n$-place symbol $f \in \Sigma_p$ a $\Sigma_t$-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 assume that $\Sigma_p$ 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 $\text{pred}$ one could use its characteristic Boolean function $f_{\text{pred}}(\text{pred}(x_1, \ldots, x_n) \Leftrightarrow f_{\text{pred}}(x_1, \ldots, x_n) = \text{true})$ 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**

<table>
<thead>
<tr>
<th>function</th>
<th>fixed values formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>int (w(node, node))</td>
<td>(edge(x_1, x_2))</td>
</tr>
<tr>
<td>node :: seqGet(seq, int)</td>
<td>(0 \leq x_2 &amp; x_2 &lt; \text{seqLen}(x_1)) &amp; node :: instance(any :: seqGet(x_1, x_2))</td>
</tr>
<tr>
<td>seq seqSub(seq, int, int)</td>
<td>(0 \leq x_2 &amp; x_2 \leq x_3 &amp; x_3 &lt; \text{seqLen}(x_1))</td>
</tr>
<tr>
<td>int / int</td>
<td>(x_2 \neq 0)</td>
</tr>
<tr>
<td>int (p_{w}(seq))</td>
<td>(\forall i((0 \leq i &amp; i &lt; \text{seqLen}(x_1)) - 1) \rightarrow) (\begin{aligned} &amp; edge(\text{node} :: \text{seqGet}(x_1, i), \ &amp; (\text{node} :: \text{seqGet}(x_1, i + 1)))\end{aligned})</td>
</tr>
</tbody>
</table>

We have adopted the requirement that \(fix_f\) be a \(\Sigma_t\)-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 \(\Sigma\) 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 \(\Sigma_{i<to} t(i)\). \(bsum\) 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 \(\mathcal{M} = (\mathcal{M}, I)\) is a partial \(\Sigma\)-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 $f \in \Sigma_p$

$$\text{fix}_f^M \subseteq \text{dom}(I(f)),$$

with the abbreviation

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

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 $\text{dom}(pf)$ for $n$-ary partial functions $pf : X^n \to Y$, $\text{dom}(pf) = \{(e_1, \ldots, e_n) \in X^n \mid pf(e_1, \ldots, e_n) \text{ is defined}\}$.

A partial structure $M = (M, I)$ 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 $\phi$ 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 structures 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 $\mathcal{M} = (M, I^M)$ be a total structure of the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$. The partial reduction of $\mathcal{M}$, denoted by $\mathcal{M}_\Sigma^p = (M, I^M_p)$, if given by

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

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

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 $\mathcal{M}_\Sigma^p$ is different from $\mathcal{N}^0$. On the other hand

**Lemma 3**

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

$$\mathcal{N}_\Sigma^p = \mathcal{M}_\Sigma^p$$

**Proof** Easy computation.

**Definition 10 (Partial Semantics)**

Let $\mathcal{M} = (M, I)$ be a partial structure over the partial signature $\Sigma = \Sigma_p \cup \Sigma_t$, $\phi$ a $\Sigma$-formula, and $\beta$ a variables assignment.

We define $\phi$ 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 $\Sigma$ be a partial signature, $\Phi$ a set of $\Sigma$-formulas and $\phi$ a single $\Sigma$-formula.

1. We say that $\phi$ is a partial logical consequence of $\Phi$, in symbols

$$\Phi \vdash_p \phi,$$

if for all partial $\Sigma$ structures $\mathcal{M}$ and all variable assignments $\beta$

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

2. We still write

$$\Phi \vdash \phi,$$

for the usual logical consequence relation where $\Sigma$ 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 $\Sigma$ be a partial structure, $\Phi$ a set of $\Sigma$-formulas and $\phi$ a single $\Sigma$-formula. Then

$$\Phi \vdash_p \phi \iff \Phi \vdash \phi$$

Proof  We assume in this proof that $\Phi$ and $\phi$ contain no free variables. The same proof works for the general case by adding $\beta$ everywhere.

Assume $\Phi \vdash \phi$ and let $\mathcal{M}$ be a partial $\Sigma$ 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 $\mathcal{M} \models \phi$. 

$\blacksquare$
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

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

and

$$\Phi \models_T \phi \iff \text{ for all models } \mathcal{M} \\
\text{ if } \mathcal{M} \models_T \Phi \text{ then } \mathcal{M} \models_T \phi$$

Then:

$$\Phi \models_T \phi \iff \Phi \models \phi$$

**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 $\Sigma$-theory, i.e. a set of $\Sigma$-sentences.

A formula $\phi$ is called total with respect to $Th$ iff

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

27
We call a formula $\varphi$ 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 $\Sigma$-theory.

Every $\Sigma_t$-formula is total with respect to $Th$.

**Proof** Obvious.

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

**Lemma 7**

Let $\Sigma$ be a partial signature, and $T$ a $\Sigma$-theory

1. If $T \vdash \varphi$ or $T \vdash \neg \varphi$ then $\varphi$ is total with respect to $T$.

   In particular if $\varphi$ is a tautology or contradictory then $\varphi$ is total.

2. If $\varphi, \psi$ are total with respect to $T$ so are $\varphi \land \psi$, $\varphi \lor \psi$, $\neg \varphi$, $\forall x \varphi$, $\exists x \varphi$.

**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 $\Sigma$ models $M$ and $N$ of $T$ such that $M^p_\Sigma = N^p_\Sigma$ and $M \models \neg \varphi$. If $N \models \varphi$ then we obtain from the assumption that $\varphi$ is total and the observation that the definition of totality is symmetrical in $M$ and $N$ that also $M \models \varphi$. A contradiction. Thus $N \models \neg \varphi$.

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 $\Sigma$. 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 $\Sigma$.

**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 $\Sigma$ if for any total $\Sigma$-sentence $\phi$

$$Th_1 \vdash \phi \iff Th_2 \vdash \phi$$

**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 $\Sigma$-structure $M$ with $M \models Th_1$ there is an extension $N$ of the partial reduction $M^p_\Sigma$ with $N \models Th_2$

and vice versa:

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

Then $Th_1$ and $Th_2$ agree on the fixed part of $\Sigma$.

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

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

1. determine whether 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 alluded 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 $\Sigma$-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:

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

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

The following definition gives a purely syntactic definition of a well-definedness predicate $wd_\phi$ for any $\Sigma$-formula $\phi$.

Definition 15
For any formula $\phi(x_1, \ldots, x_n)$ over a partial signature $\Sigma = \Sigma_p \cup \Sigma_t$ we define a $\Sigma_t$-formula $wd_\phi$ 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. $wd_{\phi_1 \land \phi_2} = wd_{\phi_1} \land wd_{\phi_2}$, $wd_{\phi_1 \lor \phi_2} = wd_{\phi_1} \land wd_{\phi_2}$, $wd_{\phi_1 \to \phi_2} = wd_{\phi_1} \land wd_{\phi_2}$
4. $wd_{\forall x \phi} = \left\{ \begin{array}{l} \forall x(\phi_0 \to wd_{\phi_1}) & \text{if } \phi \equiv \phi_0 \to \phi_1 \text{ with } \phi_0 \text{ a } \Sigma_t\text{-formula} \\ \forall xwd_\phi & \text{otherwise} \end{array} \right.$
5. $wd_{\exists x \phi} = \left\{ \begin{array}{l} \forall x(\phi_0 \to wd_{\phi_1}) & \text{if } \phi \equiv \phi_0 \land \phi_1 \text{ with } \phi_0 \text{ a } \Sigma_t\text{-formula} \\ \forall xwd_\phi & \text{otherwise} \end{array} \right.$

The formula $wd_\phi$ strongly depends on the accidental syntactic structure of $\phi$. The $wd$ formula for $\forall x(\phi_1 \to \phi_2)$ is different from the $wd$ formula for $\forall x(\neg \phi_1 \lor \phi_2)$. 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 $\phi_0$ 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 $wd_\phi$ be as in Definition 15. Then for any (total) $\Sigma$-structure $M$ and $(a_1, \ldots, a_n) \in M^n$ the following is true:

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

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

**Proof** By structural induction on the formula $\phi$.

(terms) As a preparatory step we show for any $\Sigma$-term $t$:

Let $M$ be an arbitrary (total) $\Sigma$-structure, $(a_1, \ldots, a_n) \in M^n$ with $M \models wf_t[a_1, \ldots, a_n]$.

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

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

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 interpretation of a total function $f$ is unchanged in all four structures $M$, $M^p_\Sigma$, $N^p_\Sigma$, and $N$.
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 \( wf_t = f i x_f \)) thus the value of \( f(a_1, \ldots, a_n) \)

- is the same in \( M \) and \( M^p_\Sigma \),
- is the same in \( M^p_\Sigma \) and \( N^p_\Sigma \)
  since \( M^p_\Sigma = N^p_\Sigma \)
- is the same in \( N^p_\Sigma \) and \( N \)

This guarantees:
\[
    f^M(a_1, \ldots, a_n) = f^N(a_1, \ldots, a_n)
\]

(atomic formulas) Let \( \phi = p(t_1, \ldots, t_m) \) for a predicate symbol \( p \). Assume \( M \models wd_\phi[a_1, \ldots, a_m] \) and \( M^p_\Sigma = N^p_\Sigma \). By definition \( wd_\phi = \bigwedge_{1 \leq i \leq m} wf_{t_i} \). As shown in the first paragraph of this proof \( t^M_i[\bar{a}] = t^N_i[\bar{a}] \). Thus
\[
    M \models p(t_1, \ldots, t_m)[\bar{a}] \iff (t^M_1[\bar{a}], \ldots, t^M_m[\bar{a}]) \in p^M \quad \text{semantics def.}
    \iff (t^N_1[\bar{a}], \ldots, t^N_m[\bar{a}]) \in p^N \quad \text{see above}
    \iff (t^N_1[\bar{a}], \ldots, t^N_m[\bar{a}]) \in p^N \quad \text{total predicates}
    \iff N \models p(t_1, \ldots, t_m)[\bar{a}] \quad \text{semantics def.}
\]

(propositional connectives) These cases are all straight forward. We only present that case of a disjunction in detail. We suppress the assignment \([\bar{a}]\) of free variables.

Assumption \( M \models wd_{\phi_1 \lor \phi_2} \) with \( wd_{\phi_1 \lor \phi_2} = wd_{\phi_1} \land wd_{\phi_2} \).
\[
    M \models \phi_1 \lor \phi_2 \iff M \models \phi_i \text{ for some } i \quad \text{semantics definition}
    \iff N \models \phi_i \quad \text{Ind.Hyp. using } M \models wd_{\phi_i}
    \iff N \models \phi_1 \lor \phi_2 \quad \text{semantics definition}
\]

(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 \( M \models \forall x_0 wd_{\phi_0}[\bar{a}] \) for \( \bar{a} = a_1, \ldots, a_m \).
\[
    M \models \forall x_0 \phi_0[\bar{a}] \iff M \models \phi_0[a, \bar{a}] \text{ for all } a \in M \quad \text{semantics definition}
    \iff N \models \phi_0[a, \bar{a}] \text{ for all } a \in M \quad \text{Ind.Hyp. and}
    \iff N \models \forall x_0 \phi_0[\bar{a}] \quad \text{semantics definition}
\]
Now consider the case $\phi = \forall x_0(\phi_0 \rightarrow \phi_1)(x_0, \bar{x})$ with $\phi_0$ a $\Sigma_t$-formula. Remember, we have $wd_\phi = \forall x_0(\phi_0 \rightarrow wd_{\phi_1})$.

\[ M \models \phi[a] \iff M \models \phi_0 \rightarrow \phi_1[a, \bar{a}] \text{ for all } a \in M \text{ semantics def.} \]

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

\[ \iff N \models \phi_0 \rightarrow \phi_1[a, \bar{a}] \text{ } a \in M \]

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

\[ M \models \phi[a] \iff N \models \phi[a] \]

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

\[ M \models \exists x_0 \phi_0[a] \iff M \models \phi_0[a, \bar{a}] \text{ for some } a \in M \text{ semantics def.} \]

\[ \iff N \models \phi_0[a, \bar{a}] \text{ for some } a \in M \text{ Ind.Hyp and } \]

\[ M \models wd_{\phi_0}[a, \bar{a}] \iff \]

\[ N \models \exists x_0 \phi_0[a] \]

Notice, that the induction hypothesis is applicable because initially we have $M \models wd_{\phi}[\bar{a}]$. Since $wd_{\phi} = \forall x_0 wd_{\phi_0}$, we obtain $M \models wd_{\phi_0}[a, \bar{a}]$ 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 $\phi_0$ a $\Sigma_t$-formula and case $wd_\phi = \forall x_0(\phi_0 \rightarrow wd_{\phi_1})$.

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

From $M \models \phi_0[a, \bar{a}]$
We get $N \models \phi_0[a, \bar{a}]$ since $\phi_0$ is a $\Sigma_t$-formula

From $M \models \phi_0[a, \bar{a}]$
We also get $M \models wd_{\phi_0}[a, \bar{a}]$ since $wd_{\phi} = \forall x_0(\phi_0 \rightarrow wd_{\phi_1})$
Thus by Ind.Hyp $N \models \phi_1[a, \bar{a}]$

\[ \iff N \models \phi_0 \& \phi_1[a, \bar{a}] \]

\[ \iff N \models \exists x_0(\phi_0 \& \phi_1)[\bar{a}] \]

\[ \]
Corollary 10
Let $\Sigma = \Sigma_p \cup \Sigma_t$ be a partial signature, $Th$ a $\Sigma$-theory and $wd_\phi$ 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 $\phi$ 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 $\Sigma_t$, $t_1, \ldots, t_n, t_{n+1}$ 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
$$\forall \bar{x}(f(t_1, \ldots, t_n) = t_{n+1})$$
is a total formula

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

Proof  If $\forall \bar{x}(fixf(t_1, \ldots, t_n))$ is not a tautology there is a $\Sigma$-structure $M = (M, I)$ and a variable assignment $\beta$ with $(M, \beta) \models \neg fixf(t_1, \ldots, t_n)$. We can thus find an extension $M' = (M, I')$ of $M_{\Sigma}$ that differs from $M$ only in the interpretation of the symbol $f$
$$I'(f)(a_1, \ldots, a_n) = \begin{cases} b & \text{if } a_i = t_i^{(M, \beta)} \text{ for all } i \\ I(f)(a_1, \ldots, a_n) & \text{otherwise} \end{cases}$$

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

In case $(M, \beta) \models f(t_1, \ldots, t_n) \neq t_{n+1}$ chose $b = t_{n+1}^{(M, \beta)}$. Thus $(M', \beta) \models f(t_1, \ldots, t_n) = t_{n+1}$. 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 $\Sigma$ theory axiomatized by a set of total formulas, $f$ an $n$-place function symbol in $\Sigma_t$, $t_1, \ldots, t_n, t_{n+1}$ terms that do not contain the symbol $f$, $(x_1, \ldots, x_k) = \bar{x}$ all variables occurring in $t_1$ to $t_{n+1}$. If

$$\forall \bar{x}(f(t_1, \ldots, t_n) = t_{n+1})$$

is a total formula with respect to $T$ then

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

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 tailored towards a frequently occurring 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 $P$ a partial $\Sigma$-structure such that

1. The partial reduction (Def. 9) of any $\Sigma$-model $M$ of $Th_1$ or $Th_2$ equals $P$, i.e., $M^p_\Sigma = P$,

2. There is a (total) extension $M_2$ of $P$ with $M_2 \models Th_2$

3. For every (total) extension $M_1$ of $P$ we have $M_1 \models Th_1$

then

$Th_1$ and $Th_2$ agree on the fixed part of $\Sigma$.

Proof  According to Theorem 8 we need to show

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

2. for every $\Sigma$-structure $N$ with $N \models Th_2$ there is an extension $M$ of the partial reduction $N^p_\Sigma$ with $M \models Th_1$
For the proof of 1. we note that $M_p = F$ by assumption 1 and the required extensions $N$ exists by assumption 2. For the proof of 2. we note again that $N_p = F$ by assumption 3 every total extension of $N$ 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].

$$
\begin{align*}
T(p(t_1, \ldots, t_n)) &= p(t_1, \ldots, t_n) \land \bigwedge_{i=1}^{i=n} w f_i \\
F(p(t_1, \ldots, t_n)) &= \neg p(t_1, \ldots, t_n) \land \bigwedge_{i=1}^{i=n} w f_i \\
T(\neg \phi) &= F(\phi) \\
F(\neg \phi) &= T(\phi) \\
T(\phi_1 \land \phi_2) &= T(\phi_1) \land T(\phi_2) \\
F(\phi_1 \land \phi_2) &= F(\phi_1) \lor F(\phi_2) \\
T(\phi_1 \lor \phi_2) &= T(\phi_1) \lor T(\phi_2) \\
F(\phi_1 \lor \phi_2) &= F(\phi_1) \land F(\phi_2) \\
T(\forall x \phi) &= \forall x T(\phi) \\
F(\forall x \phi) &= \exists x F(\phi) \\
T(\exists x \phi) &= \exists x T(\phi) \\
F(\exists x \phi) &= \forall x F(\phi)
\end{align*}
$$

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

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 $T$ and $F$. We repeat it here for the reader’s convenience.

**Lemma 14**

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

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

are tautologies.

Free variables in $T(\phi) \rightarrow \phi$ and $F(\phi) \rightarrow \neg \phi$ are implicitly universally quantified.

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

$$
T(\phi) \rightarrow \phi \quad \text{and} \quad F(\phi) \rightarrow \neg \phi
$$
By definition $T(\exists x \phi) = \exists x T(\phi)$. From the hypothesis we get $\exists x T(\phi) \rightarrow \exists x \phi$ and this immediately $T(\exists x \phi) \rightarrow \exists x \phi$.

Again by definition of $F$ we have $F(\exists x \phi) = \forall x F(\phi)$ while the hypothesis entails $\forall x F(\phi) \rightarrow \forall x \neg \phi$. Together we get immediately $F(\exists x \phi) \rightarrow \neg \exists x \phi$.

The remaining cases are left to the reader or we refer to [7].

Lemma 15

1. For any $\phi$ the formulas $T(\phi), F(\phi), Y(\phi)$ are total.

2. We consider two full $\Sigma$ models $M$ and $N$ of $T$ such that $M^p_\Sigma = N^p_\Sigma$.
   If $M \models \phi \land Y(\phi)$ then $N \models \phi$.

3. If $T \vdash Y(\phi)$ then $\phi$ is total with respect to $T$.

Proof

Claim (1) is proved by simultaneous induction on the complexity of $\phi$. If $\phi$ is an atomic formula $p(t_1, \ldots, t_2)$ then $T(p(t_1, \ldots, t_n)) = p(t_1, \ldots, t_n) \land \bigwedge_{i=1}^{i=n} w_{f_i}, F(p(t_1, \ldots, t_n)) = \neg p(t_1, \ldots, t_n) \land \bigwedge_{i=1}^{i=n} w_{f_i}$ 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 $Y(\phi)$ for atomic $\phi$. If $\phi = \phi_1 \land \phi_2$ we need to show that $T(\phi) = T(\phi_1) \land T(\phi_2)$ is total with respect to $T$. By induction hypothesis $T(\phi_1)$ and $T(\phi_2)$ are total thus also $T(\phi_1) \land T(\phi_2)$. Along the same lines totality of $F(\phi_1 \land \phi_2)$ and $Y(\phi_1 \land \phi_2)$ is shown. Also the induction step for $\phi = \phi_1 \lor \phi_2$ does not pose any problem. In the step case $\phi = \neg \phi_0$ we have by simultaneous induction the totality if $T(\phi_0), F(\phi_0),$ and $Y(\phi_0)$ at our disposal. Since $T(\neg \phi_0) = F(\phi_0)$ and vice versa $F(\neg \phi_0) = T(\phi_0)$ 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 $\phi$.

atomic case  If $\phi$ is the atomic formula $p(t_1, \ldots, t_2)$ then

\[
Y(p(t_1, \ldots, t_n)) = T(p(t_1, \ldots, t_n) \lor F(p(t_1, \ldots, t_n))
\]

\[
= (p(t_1) \land \bigwedge_{i=1}^{i=n} w_{f_i}) \lor (\neg p(t_1) \land \bigwedge_{i=1}^{i=n} w_{f_i})
\]

\[
\Leftrightarrow (p(t_1) \lor \neg p(t_1)) \land \bigwedge_{i=1}^{i=n} w_{f_i}
\]

\[
\Leftrightarrow \bigwedge_{i=1}^{i=n} w_{f_i}
\]

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

37
**conjunction** We start from $\mathcal{M} \models (\phi_1 \land \phi_2) \land \gamma(\phi_1 \land \phi_2)$. By Lemma 14 we must have $\mathcal{M} \models T(\phi_1 \land \phi_2)$ since $\mathcal{M} \models F(\phi_1 \land \phi_2)$ would lead to the contradiction $\mathcal{M} \models (\phi_1 \land \phi_2) \land \neg(\phi_1 \land \phi_2)$. This entails $\mathcal{M} \models \phi_i \land T(\phi_i)$ and thus $\mathcal{M} \models \phi_i \land \gamma(\phi_i)$ for $i = 1, 2$. Induction hypothesis yields $\mathcal{N} \models \phi_i \land \gamma(\phi_i)$, as desired.

**disjunction** Absolutely analogous to the conjunction case.

**negation** We start from $\mathcal{M} \models \neg \phi \land \gamma(\neg \phi)$. By Lemma 14 we get $\mathcal{M} \models T(\neg \phi)$ which yields by definition of $T$ that $\mathcal{M} \models \neg \phi \land F(\phi)$. By item (1) of this lemma we get $\mathcal{N} \models F(\phi)$ and thus also $\mathcal{N} \models \gamma(\phi)$. 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 $\mathcal{M} \models \forall x \phi \land \gamma(\forall x \phi)$. Following the pattern from previous cases we argue that we must have $\mathcal{M} \models T(\forall x \phi)$. Since $\mathcal{M} \models F(\forall x \phi)$ would produce the contradiction $\mathcal{M} \models (\forall x \phi) \land (\neg \forall x \phi)$. By definition of $T$ we get $\mathcal{M} \models \forall x T(\phi)$. For any element $a$ in the common universe of $\mathcal{M}$ and $\mathcal{N}$ we get $\mathcal{M} \models \phi[a]$ and $\mathcal{M} \models T(\phi)[a]$. 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 $P\text{Subst}(\phi)$ we denote the formula that arises from $\phi$ by substituting

- every positive occurrence of an atomic subformula $p(t_1, \ldots, t_n)$ in $\phi$ by $\bigwedge_{i=1}^{i=n}wf_{t_i} \land p(t_1, \ldots, t_n)$
- every negative occurrence of an atomic subformula $p(t_1, \ldots, t_n)$ in $\phi$ by $\bigwedge_{i=1}^{i=n}wf_{t_i} \rightarrow p(t_1, \ldots, t_n)$.

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

- every negative occurrence of an atomic subformula $p(t_1, \ldots, t_n)$ in $\phi$ is substituted by $\bigwedge_{i=1}^{i=n}wf_{t_i} \land p(t_1, \ldots, t_n)$
- every positive occurrence of an atomic subformula $p(t_1, \ldots, t_n)$ in $\phi$ is substituted by $\bigwedge_{i=1}^{i=n}wf_{t_i} \rightarrow p(t_1, \ldots, t_n)$.

An occurrence 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 occurrences within a formula.

**Lemma 16**
The following equivalences are tautologies

1. \( P\text{Subst}(\phi_1 \land \phi_2) \iff P\text{Subst}(\phi_1) \land P\text{Subst}(\phi_2) \)
2. \( P\text{Subst}^- (\phi_1 \land \phi_2) \iff P\text{Subst}^- (\phi_1) \land P\text{Subst}^- (\phi_2) \)
3. \( P\text{Subst}(\phi_1 \lor \phi_2) \iff P\text{Subst}(\phi_1) \lor P\text{Subst}(\phi_2) \)
4. \( P\text{Subst}^- (\phi_1 \lor \phi_2) \iff P\text{Subst}^- (\phi_1) \lor P\text{Subst}^- (\phi_2) \)
5. \( P\text{Subst}^- (!\phi) \iff !P\text{Subst}^- (\phi) \)
6. \( P\text{Subst}^- (\forall x \phi) \iff \forall x \ P\text{Subst}(\phi) \)
7. \( P\text{Subst}^- (\exists x \phi) \iff \exists x \ P\text{Subst}^- (\phi) \)
8. \( P\text{Subst}^- (\neg \phi) \iff \neg P\text{Subst} (\phi) \)
9. \( P\text{Subst}^- (\neg \exists x \phi) \iff P\text{Subst}(\forall x \neg \phi) \)
10. \( P\text{Subst}^- (\neg \neg \phi) \iff P\text{Subst} (\phi) \)

**Proof** A really formal proof would need a notation system for occurrences 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

\[
P\text{Subst}^- (\neg (\phi_1 \land \phi_2)) \iff \neg P\text{Subst}^- (\phi_1 \land \phi_2) \\
\iff \neg (P\text{Subst}^- (\phi_1) \land P\text{Subst}^- (\phi_2)) \\
\iff \neg P\text{Subst}^- (\phi_1) \lor \neg P\text{Subst}^- (\phi_2) \\
\iff P\text{Subst}^- (\neg \phi_1) \lor P\text{Subst}^- (\neg \phi_2)
\]

It is tempting to conjecture that for a tautology \( \phi \) also \( P\text{Subst}(\phi) \) is a tautology. But, \( p(t) \lor \neg p(t) \) is always a tautology while \( P\text{Subst}(p(t) \lor \neg p(t)) = (p(t) \land w_{f_1}) \lor (\neg p(t)) \land w_{f_1} \iff w_{f_1} \) is in general not.

**Lemma 17**
Let \( \phi \) be a \( \Sigma \)-formula then

1. \( T(\phi) \iff P\text{Subst}(\phi) \)
2. $\mathcal{F}(\phi) \Leftrightarrow PSubst(\neg \phi)$

**Proof** The proof proceeds by simultaneous induction on the complexity of $\phi$.

**atomic case**

\[
\begin{align*}
    \mathcal{T}(p(t_1, \ldots, t_n)) &= \bigwedge_{i=1}^{i=n} w f_t \land p(t_1, \ldots, t_n) \\
    &= PSubst(p(t_1, \ldots, t_n)) \\
    \mathcal{F}(p(t_1, \ldots, t_n)) &= \bigwedge_{i=1}^{i=n} w f_t \land \neg p(t_1, \ldots, t_n) \\
    &= \neg (\bigwedge_{i=1}^{i=n} w f_t \rightarrow p(t_1, \ldots, t_n)) \\
    &= PSubst(\neg p(t_1, \ldots, t_n))
\end{align*}
\]

**conjunction**

\[
\begin{align*}
    \mathcal{T}(\phi_1 \land \phi_2) &= \mathcal{T}(\phi_1) \land \mathcal{T}(\phi_2) \quad \text{Def. } \mathcal{T} \\
    &= PSubst(\phi_1) \land PSubst(\phi_2) \quad \text{Ind.Hyp.} \\
    &= PSubst(\phi_1 \land \phi_2) \quad \text{property of } Psubst \\
    \mathcal{F}(\phi_1 \land \phi_2) &= \mathcal{F}(\phi_1) \lor \mathcal{F}(\phi_2) \quad \text{Def. } \mathcal{F} \\
    &= PSubst(\neg \phi_1) \lor PSubst(\neg \phi_2) \quad \text{Ind.Hyp.} \\
    &= PSubst(\neg \phi_1 \lor \neg \phi_2) \quad \text{property of } Psubst \\
    &= PSubst(\neg(\phi_1 \land \phi_2)) \quad \text{property of } Psubst
\end{align*}
\]

**disjunction** Analogous to the conjunction case.

**negation**

\[
\begin{align*}
    \mathcal{T}(\neg \phi) &= \mathcal{F}(\phi_1) \quad \text{Def. } \mathcal{T} \\
    &= PSubst(\neg \phi) \quad \text{Ind.Hyp.} \\
    \mathcal{F}(\neg \phi) &= \mathcal{T}(\phi_1) \quad \text{Def. } \mathcal{F} \\
    &= PSubst(\phi) \quad \text{Ind.Hyp.} \\
    &= PSubst(\neg \phi) \quad \text{property of } Psubst
\end{align*}
\]

**quantification**

\[
\begin{align*}
    \mathcal{T}(\forall x \phi) &= \forall x \mathcal{T}(\phi) \quad \text{Def. } \mathcal{T} \\
    &= \forall x PSubst(\phi) \quad \text{Ind.Hyp.} \\
    &= PSubst(\forall x \phi) \quad \text{property of } Psubst \\
    \mathcal{F}(\forall x \phi) &= \exists x \mathcal{F}(\phi) \quad \text{Def.} \\
    &= \exists x PSubst(\neg \phi) \quad \text{Ind.Hyp.} \\
    &= PSubst(\neg \exists x \phi) \quad \text{property of } Psubst
\end{align*}
\]

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) $\Sigma$-structure for a partial signature $\Sigma$. A 3-valued structure $\mathcal{M}^3 = (M^3, I^3)$ is defined by:

$M^3 = M \cup \{\bot\}, \bot \notin M$

$I^3(f)(a_1, \ldots, a_n) = \begin{cases} I(f)(a_1, \ldots, a_n) & \text{if } (a_1, \ldots, a_n) \in M^n \text{ and } f \in \Sigma_t \text{ or } M \models \text{fix}_f[a_1, \ldots, a_n] \\ \bot & \text{otherwise} \end{cases}$

$I^3(p)(a_1, \ldots, a_n) = \begin{cases} 1 & \text{if } (a_1, \ldots, a_n) \in M^n \text{ and } M \models p[a_1, \ldots, a_n] \\ \frac{1}{2} & a_i = \bot \text{ for some } i \\ 0 & \text{if } (a_1, \ldots, a_n) \in M^n \text{ and } M \models \neg p[a_1, \ldots, a_n] \end{cases}$

As a consequence of this definition we get for all function symbols $f$ in $\Sigma$ 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:

$I^3(\phi_1 \land \phi_2) = \min\{I^3(\phi_1), I^3(\phi_2)\}$

$I^3(\phi_1 \lor \phi_2) = \max\{I^3(\phi_1), I^3(\phi_2)\}$

$I^3(\neg \phi) = 1 - I^3(\phi)$

$I^3(\phi_1 \rightarrow \phi_2) = I^3(\neg \phi_1 \lor \phi_2)$

$I^3(\forall x \phi) = \min\{I^3(\phi[a]) \mid a \in M\}$

$I^3(\exists x \phi) = \max\{I^3(\phi[a]) \mid a \in M\}$

The minimum and maximum 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**

1. $I^3(\frac{1}{2} > 0 \lor 1 > 0) = 1$
2. $I^3(\frac{1}{2} > 0 \lor \frac{1}{2} \leq 0) = \frac{1}{2}$
3. $I^3(\frac{1}{2} > 0 \rightarrow 1 > 0) = 1$
4. $I^3(\forall x(\frac{1}{x} \ast x = x)) = \frac{1}{2}$
5. $I^3(\exists x(\frac{1}{x} \ast x = x)) = 1$

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 $\Sigma$-structure, $t$ a term, $\phi$ a $\Sigma$-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. $\mathcal{M} \models T(\phi)[a_1, \ldots, a_n]$ iff $I^3(\phi[a_1, \ldots, a_n]) = 1$
3. $\mathcal{M} \models F(\phi)[a_1, \ldots, a_n]$ iff $I^3(\phi[a_1, \ldots, a_n]) = 0$
4. $\mathcal{M} \models Y(\phi)[a_1, \ldots, a_n]$ 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 $\mathcal{M} \models fix_f[I^M(t_1)[\bar{a}], \ldots, I^M(t_n)[\bar{a}]]$ yields $I(t_i)(a_1, \ldots, a_n) = I^3(t_i)(a_1, \ldots, a_n)$ for all $i$. Now

$I(t)(\bar{a}) = I(f)(I(t_1)(\bar{a}), \ldots, I(t_n)(\bar{a}))$ 

$= I^3(f)(I(t_1)(\bar{a}), \ldots, I(t_n)(\bar{a}))$ 

$= I^3(f)(I^3(t_1)(\bar{a}), \ldots, I^3(t_n)(\bar{a}))$ 

$= I^3(t)(\bar{a})$ 

**Items 2 and 3** The proof proceeds by simultaneous induction on the complexity of $\phi$.

**Atomic formulas**

$\mathcal{M} \models T(p(\bar{t})[\bar{a}]$ $\iff$ $\mathcal{M} \models (p(\bar{t}) \land \bigwedge_{i=1}^{n} wf_{t_i})[\bar{a}]$ 

$\iff$ $\mathcal{M} \models p[\bar{b}]$ and $b_i = I(t_i)[\bar{a}] = I^3(t_i)[\bar{a}]$ 

$\iff$ $I^3(p[\bar{b}]) = 1$ 

$\iff$ $I^3(p(\bar{t})[\bar{a}]) = 1$ 

**Def. $T$** 

**Item 1**

$\mathcal{M} \models F(p(\bar{t})[\bar{a}]$ $\iff$ $\mathcal{M} \models (\neg p(\bar{t}) \land \bigwedge_{i=1}^{n} wf_{t_i})[\bar{a}]$ 

$\iff$ $\mathcal{M} \models \neg p[\bar{b}]$ and $b_i = I(t_i)[\bar{a}] = I^3(t_i)[\bar{a}]$ 

$\iff$ $I^3(p[\bar{b}]) = 0$ 

$\iff$ $I^3(p(\bar{t})[\bar{a}]) = 0$ 

**Def. $F$** 

**Item 1**

**Conjunction**

$\mathcal{M} \models F(p(\bar{t})[\bar{a}]$ $\iff$ $\mathcal{M} \models (p(\bar{t}) \land \bigwedge_{i=1}^{n} wf_{t_i})[\bar{a}]$ 

$\iff$ $\mathcal{M} \models p[\bar{b}]$ and $b_i = I(t_i)[\bar{a}] = I^3(t_i)[\bar{a}]$ 

$\iff$ $I^3(p[\bar{b}]) = 1$ 

$\iff$ $I^3(p(\bar{t})[\bar{a}]) = 1$ 

**Def. $I^3$**
\[M \models T(\phi_1 \land \phi_2)[\bar{a}] \iff M \models (T(\phi_1) \land T(\phi_2))[\bar{a}] \quad \text{Def. } T\]
\[\iff I^3(\phi_1[\bar{a}]) = 1 \text{ and } I^3(\phi_2[\bar{a}]) = 1 \quad \text{I.Hyp.}\]
\[\iff I^3((\phi_1 \land \phi_2)[\bar{a}]) = 1 \quad \text{Def. } I^3\]

\[M \models F(\phi_1 \land \phi_2)[\bar{a}] \iff M \models (F(\phi_1) \lor F(\phi_2))[\bar{a}] \quad \text{Def. } F\]
\[\iff M \models F(\phi_1)[\bar{a}] \text{ or } M \models F(\phi_2)[\bar{a}] \quad \lor\]
\[\iff I^3(\phi_1[\bar{a}]) = 0 \text{ or } I^3(\phi_2[\bar{a}]) = 0 \quad \text{I.Hyp.}\]
\[\iff I^3((\phi_1 \land \phi_2)[\bar{a}]) = 0 \quad \text{Def. } I^3\]

disjunction analogous to conjunction

\[M \models T(\neg \phi)[\bar{a}] \iff M \models F(\phi)[\bar{a}] \quad \text{Def. } T\]
\[\iff I^3(\phi[\bar{a}]) = 0 \quad \text{Ind.Hyp.}\]
\[\iff I^3(\neg \phi[\bar{a}]) = 1 \quad \text{Def. } I^3\]

The case \(M \models F(\neg \phi) \iff I^3(\neg \phi) = 0\) follow vice versa.

\[M \models T(\forall x_0 \phi)[\bar{a}] \iff M \models (\forall x_0 T(\phi))[\bar{a}] \quad \text{Def. } T\]
\[\iff M \models T(\phi)[a_0, \bar{a}] \text{ for all } a_0 \in M \quad \text{Def. } \forall\]
\[\iff I^3(\phi[a_0, \bar{a}]) = 1 \text{ for all } a_0 \in M \quad \text{Ind.Hyp.}\]
\[\iff I^3(\forall x_0 \phi[\bar{a}]) = 1 \quad \text{Def. } I^3\]

We leave the remaining case \(T(\exists x_0 \phi), F(\forall x_0 \phi), F(\exists x_0 \phi)\) to the reader.

**Item 4** follows directly from 2 and 3.

### 2.5 Examples

**Example 20**

Let us compute \(wd_\phi\) for

\[\phi(s_1, s_2) \equiv (s_1 = s_2 \leftrightarrow \text{seqLen}(s_1) = \text{seqLen}(s_2) \land \forall \text{int } i(0 \leq i < \text{seqLen}(s_1) \rightarrow \text{any :: seqGet}(s_1, i) = \text{any :: seqGet}(s_2, i)))\]

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.

\[wd_\phi(s_1, s_2) = (\text{true} \land \text{true} \land \forall \text{int } i(0 \leq i < \text{seqLen}(s_1) \rightarrow 0 \leq i \land i < \text{seqLen}(s_1) \land i < \text{seqLen}(s_2)))\]
\[\iff (\forall \text{int } i(0 \leq i < \text{seqLen}(s_1) \rightarrow 0 \leq i \land i < \text{seqLen}(s_1) \land i < \text{seqLen}(s_2)))\]

43
Further

\[wd_{\text{Seq}} s_1, s_2 \phi = \forall \text{Seq } s_1, s_2 \forall \text{int } i (0 \leq i & i < \text{seqLen}(s_1) \rightarrow 0 \leq i & i \leq \text{seqLen}(s_1) & i \leq \text{seqLen}(s_2)) \leftrightarrow \forall \text{Seq } s_1, s_2 \forall \text{int } i (0 \leq i & i < \text{seqLen}(s_1) \rightarrow i \leq \text{seqLen}(s_2))\]

If \(Th_{\text{seq}}\) is the overspecified theory of sequences given Section 2.7, then \(Th_{\text{seq}} \not\vdash \forall s_1, s_2 \phi\). 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 \(\text{seqLen}(s_1) = \text{seqLen}(s_2)\) stated in the first conjunct. This emphasizes the point the \(wd\) 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 \(\psi\) is total with respect to \(Th\) and \(Th \models \psi \leftrightarrow \psi'\) then also \(\psi'\) 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 \(\phi\) to an equivalent formula \(\phi'\) would yield \(Th_{\text{seq}} \vdash \forall s_1, s_2 \phi'\):

\[\phi'(s_1, s_2) \equiv (s_1 = s_2 \leftrightarrow \text{seqLen}(s_1) = \text{seqLen}(s_2) & \forall \text{int } i (0 \leq i & i < \text{seqLen}(s_1) & i < \text{seqLen}(s_2) \rightarrow \text{any} :: \text{seqGet}(s_1, i) = \text{any} :: \text{seqGet}(s_2, i)))\]

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

\[\phi(s_1, s_2) \equiv (s_1 = s_2 \leftrightarrow \text{seqLen}(s_1) = \text{seqLen}(s_2) & \forall \text{int } i (0 \leq i < \text{seqLen}(s_1) \rightarrow \text{any} :: \text{seqGet}(s_1, i) = \text{any} :: \text{seqGet}(s_2, i)))\]

For conciseness we write \(s[i]\) for \(\text{any} :: \text{seqGet}(s, i)\), \(\text{Len}\) for \(\text{seqLen}\), and \(\forall i \exists i\) for \(\forall \text{int } i \exists \text{int } i\).
\( \mathcal{T}(\phi) = (\mathcal{T}(s_1 \neq s_2) \lor \mathcal{T}(\text{Len}(s_1) = \text{Len}(s_2) \land \forall i(\ldots))) \land \\
(\mathcal{T}(s_1 = s_2) \lor \mathcal{T}(\neg(\text{Len}(s_1) = \text{Len}(s_2) \land \forall i(\ldots)))) \\
= (s_1 \neq s_2 \lor (\text{Len}(s_1) = \text{Len}(s_2) \land \mathcal{T}(\forall i(\ldots))) \land \\
(s_1 = s_2 \lor (\text{Len}(s_1) \neq \text{Len}(s_2) \lor \mathcal{T}(\neg\forall i(\ldots)))) \\
= (s_1 \neq s_2 \lor (\text{Len}(s_1) = \text{Len}(s_2) \land \\
\forall i(0 \leq i \land i < \text{Len}(s_1) \rightarrow \mathcal{T}(s_1[i] = s_2[i]))) \land \\
(s_1 = s_2 \lor (\text{Len}(s_1) \neq \text{Len}(s_2) \land \\
\exists i(0 \leq i \land i < \text{Len}(s_1) \land \mathcal{F}(s_1[i] = s_2[i]))) \\
= (s_1 \neq s_2 \lor (\text{Len}(s_1) = \text{Len}(s_2) \land \\
\forall i(0 \leq i \land i < \text{Len}(s_1) \rightarrow \\
(s_1[i] = s_2[i]) \land 0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2))) \land \\
(s_1 = s_2 \lor (\text{Len}(s_1) \neq \text{Len}(s_2) \land \\
\exists i(0 \leq i \land i < \text{Len}(s_1) \land \\
\exists i(0 \leq i \land i < \text{Len}(s_1) \land s_1[i] \neq s_2[i]))) \\
= \text{true} \land \\
s_1 \neq s_2 \rightarrow (\text{Len}(s_1) \neq \text{Len}(s_2)) \land \\
\exists i(0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2) \land s_1[i] \neq s_2[i]))
\[ F(\phi) = F(s_1 \neq s_2 \lor (\text{Len}(s_1) = \text{Len}(s_2) \land \forall i(\ldots))) \lor F(s_1 = s_2 \lor \neg(\text{Len}(s_1) = \text{Len}(s_2) \land \forall i(\ldots))) \]

\[ = (F(s_1 \neq s_2) \land \neg
F(\forall i(0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i]))) \lor
(s_1 \neq s_2 \land \neg\exists i F(0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i]))) \lor
(s_1 \neq s_2 \land \neg\exists i F(0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i])) \lor
\]

\[ = (s_1 = s_2 \land
\text{Len}(s_1) \neq \text{Len}(s_2) \lor
\exists i F(0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i])) \lor
(s_1 \neq s_2 \land \neg\exists i \text{Len}(s_1) = \text{Len}(s_2) \lor
\exists i (0 \leq i \land i < \text{Len}(s_1) \land \forall i(\neg(\exists i(0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2))) \lor
(s_1 \neq s_2 \land
\text{Len}(s_1) = \text{Len}(s_2) \land
\forall i(0 \leq i \land i < \text{Len}(s_1) \rightarrow
s_1[i] = s_2[i]) \land 0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2)))) \lor
\]

\[ = \text{false} \lor
(s_1 \neq s_2 \land
\text{Len}(s_1) = \text{Len}(s_2) \land
\forall i(0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i])) \]
\( \mathcal{Y}(\phi) = \mathcal{T}(\phi) \lor \mathcal{F}(\phi) \)

\[
\begin{align*}
    \mathcal{T}(\phi) &= (s_1 = s_2 \lor \text{Len}(s_1) \neq \text{Len}(s_2) \lor \exists i (\ldots)) \\
    \lor \quad & (s_1 \neq s_2 \land \text{Len}(s_1) = \text{Len}(s_2) \land \forall (\ldots)) \\
    \iff \quad & \text{true} \land \text{true} \land \\
    s_1 &= s_2 \lor \text{Len}(s_1) \neq \text{Len}(s_2) \\
    \exists i (0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2) \land s_1[i] \neq s_2[i]) \lor \\
    \forall i (0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i]) \\
    \iff \quad & (s_1 = s_2 \land (\ldots \lor \ldots)) \lor \\
    s_1 &= s_2 \land (\text{false} \lor \text{false} \lor \text{true}) \lor \\
    (s_1 \neq s_2 \land (\text{false} \lor \text{Len}(s_1) \neq \text{Len}(s_2) \lor \\
    \exists i (0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2) \land s_1[i] \neq s_2[i]) \lor \\
    \forall i (0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i]))) \\
    \iff \quad & s_1 = s_2 \lor \\
    s_1 &= s_2 \land (\text{Len}(s_1) = \text{Len}(s_2) \land \\
    \forall i (0 \leq i \land i < \text{Len}(s_1) \land i < \text{Len}(s_2) \rightarrow s_1[i] = s_2[i])) \\
    \rightarrow \quad & \forall i (0 \leq i \land i < \text{Len}(s_1) \rightarrow s_1[i] = s_2[i])) \\
    \iff \quad & s_1 = s_2 \lor (s_1 \neq s_2 \land \text{true}) \\
    \iff \quad & \text{true}
\end{align*}
\]

**Example 22**

Let us compute \( w_d \phi \) for

\[
\phi \equiv (\forall \text{node } m; (m \neq \text{start} \rightarrow (\exists \text{Seq } s; (\text{fwpath}(s) \& \\
    \text{node :: seqGet}(s, 0) = \text{start} \& \\
    \text{node :: seqGet}(s, \text{seqLen}(s) - 1) = m \& \\
    \text{pw}(s) = d(m)))))
\]

47
This is the first claim (1.9) of the Bellman-Ford-Lemma. We compute $wd_\phi$ using the fixed values formulas from Example 2.

$$wd_\phi = \forall node \text{ } m(m \neq \text{start} \rightarrow \forall Seq \text{ } s; (\text{fwpath}(s) \rightarrow \& 0 \leq 0 < \text{seqLen}(s) \& \text{node} :: \text{instance(any :: seqGet(s,0))} \& 0 \leq \text{seqLen}(s) - 1 < \text{seqLen}(s) \& \& \text{node} :: \text{instance(any :: seqGet}(s,\text{seqLen}(s) - 1)) \& \& \forall i(0 \leq i < \text{seqLen}(s) \rightarrow \text{edge(node :: seqGet}(x1,i),(\text{node :: seqGet}(x1,i+1)))) \& \& \text{node} :: \text{instance}(\text{node :: seqGet}(x1,i))))$$

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

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

then $Th_{WG} \vdash wd_{\phi'}$ can be proved and we know that $\phi'$, und thus also $\phi$ 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 $\Sigma_{WG}$ 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 $fix_{w(x_1,x_2)} = edge(x_1,x_2)$. 

48
Th_{WG}^1 is given by the axioms
\[ \forall \text{node } n; (\text{edge}(n,n)) \] (1.1)
\[ \forall \text{node } n; (\forall \text{node } m; (\text{edge}(n,m) \rightarrow w(n,m) > 0)) \] (1.2)

Th_{WG}^2 on the other hand is given by axiom (1.1) and
\[ \forall \text{node } m; (\forall \text{node } n; (w(n,m) > 0)) \] (1.2a)

It is fairly obvious that Th_{WG}^1 and Th_{WG}^2 agree on the fixed part of \( \Sigma_{WG} \).

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 \( \Sigma_0 \) that contains at least the type \( \text{int} \). As usual we consider the interpreted semantics, i.e. in any model \( M = (M,I) \) of \( T_0 \) the interpretation of \( \text{int} \) is (precisely) the set of integers and the arithmetic operations \( I(+) \), \( I(*) \), etc have their usual mathematical meaning. The signature is extended to \( \Sigma_{fact} = \Sigma_0 \cup \{ \text{fact} \} \) by a unary function symbol \( \text{fact} \) with fixed valued formula \( \text{fix}_{\text{fact}}(x) \equiv x \geq 0 \).

We want to extend \( T_0 \) to axiomatize the following partial \( \Sigma_{fact} \)-structure \( F_p \):

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

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

\( T_{fact}^1 \) adds the axioms
\[ \text{fact}(0) = 1 \] (2.4)
\[ \forall \text{int } z (z > 0 \rightarrow \text{fact}(z) = z \ast \text{fact}(z-1)) \] (2.5)

while \( T_{fact}^2 \) adds the axioms
\[ \text{fact}(0) = 1 \] (2.6)
\[ \forall \text{int } z (z \neq 0 \rightarrow \text{fact}(z) = z \ast \text{fact}(z-1)) \] (2.7)

It is not hard to see that

For any \( \Sigma_{fact} \)-model \( M \) of \( T_{fact}^1 \) or \( T_{fact}^2 \) : \( M_{\Sigma_{fact}}^\pi = \mathcal{F}_p \) (2.8)

There is a (total) extension \( M^* \) of \( \mathcal{F}_p \) with \( M^* \models T_{fact}^1 \land T_{fact}^2 \) (2.9)

Let \( \mathcal{M} \) be an arbitrary (total) \( \Sigma_{fact} \)-structure, then \( \mathcal{M} \models T_{fact}^1 \) (2.10)
For $M^* = (\mathbb{Z}, I^*)$ define $I^*(\text{fact})(i) = 0$ for all $i < 0$. Otherwise the claims (2.8), (2.9), and (2.10) are easily verified. By Lemma 13 $T_{\text{fact}}^1$ and $T_{\text{fact}}^2$ agree on the fixed part of $\Sigma_{\text{fact}}$. Thus the same total formulas are derivable in both theories.

As a third possibility let $T_{\text{fact}}^3$ be the theory obtained by adding the following axiom to $T_{\text{fact}}^1$:

$$\forall z (\text{fact}(z) \neq 0)$$

Then $T_{\text{fact}}^3$ is inconsistent since a model would have to satisfy

$$\text{fact}(-1) = (-1) \ast \text{fact}(-2) = (-1) \ast (-2) \ast \text{fact}(-3) = (-1) \ast (-2) \ast (-3) \ast \text{fact}(-4) = \cdots$$

which is impossible unless $\text{fact}(i) = 0$ for all $i < 0$.

\textbf{Example 25 (Choose Operator)}

Let $\Sigma_\varepsilon$ be a signature that contains the sorts $\text{elem}, \text{set}$, a constant $\emptyset$ of sort $\text{set}$ and the binary relation $\in (\text{elem}, \text{set})$. The $\Sigma_\varepsilon$-theory $T_\varepsilon$ consists of the axioms

$$\forall \text{elem } x (\neg x \in \emptyset)$$

$$\forall \text{set } x \forall \text{set } y (x = y \leftrightarrow \forall \text{elem } u (u \in x \leftrightarrow u \in y))$$

$\Sigma_{\text{ch}}$ extends $\Sigma_\varepsilon$ by a new function symbol $\text{choose} : \text{set} \rightarrow \text{elem}$ with the fixed value formulas $\text{fix}_{\text{choose}}(x) \equiv x \neq \emptyset$. We propose two extension $T_\varepsilon^1$ and $T_\varepsilon^2$ of $T_\varepsilon$. $T_\varepsilon^1$ is obtained by adding the axiom

$$\forall \text{set } s (s \neq \emptyset \rightarrow \text{choose}(s) \in s)$$

$T_\varepsilon^2$ is obtained by adding the axiom

$$\forall \text{set } s (\text{choose}(s) \in s)$$

$T_\varepsilon^2$ is inconsistent since $T_\varepsilon^2 \vdash \text{choose}(\emptyset) \in \emptyset$ which contradicts (2.12).

One the other hand $T_\varepsilon^1 \vdash \text{choose}(\emptyset) \notin \emptyset$ and

$T_\varepsilon^1 \vdash \forall \text{set } s (\text{choose}(s) \in s \rightarrow \text{wd}_{\text{choose}(s) \in s}(s))$ since $T_\varepsilon^1 \vdash \text{choose}(s) \in s \rightarrow s \neq \emptyset$.

50
Example 26

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

\[
\begin{align*}
\phi_0 & \equiv f(x) = x \lor f(-x) = -x \\
\phi_n & \equiv \phi_{n-1} \land (f(x + n) = x + n \lor f(-x - n) = -x - n)
\end{align*}
\]

Well-definedness of \(\phi_n\) is investigated with respect to the \(\Sigma_{bm}\)-theory \(T_{bm}\) given by the usual theory of integers as far as the signature \(\Sigma_{bm}\) is concerned plus the axiom \(\forall x (x \geq 0) \rightarrow f(x) = x\). 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

\[
\begin{align*}
wd(\phi_0) & \iff x = 0 \\
w(\phi_1) & \iff x = 0 \land x = 1 \\
w(\phi_n) & \iff false \text{ for all } n \geq 1
\end{align*}
\]

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

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

\[
\begin{align*}
Y(\phi_0) & \iff_{T_{bm}} x \geq 0 \lor -x \geq 0 \\
& \iff_{T_{bm}} true
\end{align*}
\]

Likewise \(T_{bm} \vdash Y(\phi_n) \iff true\) 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 \(\alpha \subseteq any\) (\(\alpha\) is a subtype of \(any\)) for all \(\alpha \in Types\).

The signature \(\Sigma_{seq}\) of a theory of sequences contains the additional type
$Seq$ and the following function symbols

**Constructors**

- $seqEmpty : \rightarrow Seq$
- $seqSingleton : \text{any} \rightarrow Seq$
- $seqConcat : Seq, Seq \rightarrow Seq$
- $seqSub : Seq, int, int \rightarrow Seq$
- $seqReverse : Seq \rightarrow Seq$

**Observers**

- $alpha :: seqGet : Seq, int \rightarrow alpha$
- $seqLen : Seq \rightarrow int$

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

<table>
<thead>
<tr>
<th>function</th>
<th>fixed values formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$seqSub(x_1, x_2, x_3)$</td>
<td>$0 \leq x_2 \leq x_3 &lt; seqLen(x_1)$</td>
</tr>
</tbody>
</table>
| $alpha :: seqGet(x_1, x_2)$ | $0 \leq x_2 < seqLen(x_1) \land$
|                      | $alpha :: instance(any :: seqGet(x_1, x_2))$               |

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 $\Sigma^0_{seq} = \{seqEmpty, seqSingleton, seqConcat, seqLen, int :: seqGet, any :: seqGet \}$. $Th^1_{seqC}$ 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 $SQ^0$, the data type of finite sequence in the signature $\Sigma^0_{seq}$.

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 \( U = \mathbb{Z} \cup A \cup \text{Seq} \) with
   - \( \mathbb{Z} \) as usual
   - \( A \) an infinite set that serves as the interpretation of the sort any
   - \( \text{Seq} = \{ \langle a_0, \ldots, a_n \rangle \mid n \in \mathbb{N} \text{ and } a_i \in A \} \)

2. \( I(\text{seqEmpty}) = \langle \rangle \)

3. \( I(\text{seqSingleton})(a) = \langle a \rangle \)

4. \( I(\text{seqConcat})(\langle a_0, \ldots, a_n \rangle, \langle b_0, \ldots, b_m \rangle) = \langle a_0, \ldots, a_n, b_0, \ldots, b_m \rangle \)

5. \( I(\text{seqLen})(\langle a_0, \ldots, a_n \rangle) = n + 1 \)

6. \( I(\text{any}::\text{seqGet})(\langle a_0, \ldots, a_n \rangle, i) = \begin{cases} a_i & \text{if } 0 \leq i \leq n \\ \text{undef} & \text{otherwise} \end{cases} \)

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

We want to find a first-order axiomatisation \( Th_{\text{seqC}} \) of \( \mathcal{SQ}^0 \). Later on we will compare the axiomatisation \( Th_{\text{seqC}} \), with the theory \( Th_{\text{seq}} \), and its subtheories \( Th_{\text{seq}}^0, Th_{\text{seq}}^1 \), embodied in the taclets in Section 4.2.
\[
\forall \text{seq } s_1 \forall \text{seq } s_2 (s_1 = s_2 \leftrightarrow \text{seqLen}(s_1) = \text{seqLen}(s_2)) \quad \text{(SeqAxiomC 1)}
\]
\[
\forall \text{int } i (0 \leq i \& i < \text{seqLen}(s_1) \rightarrow \forall \text{any } x : : \text{seqGet}(s_1, i) = \text{any } : : \text{seqGet}(s_2, i)) \quad \text{(SeqAxiomC 2)}
\]
\[
\forall \text{seq } s_1 \forall \text{seq } s_2 \forall \text{int } i (0 \leq i \& i < \text{seqLen}(s_1) + \text{seqLen}(s_2) \rightarrow \forall \text{any } x (\text{beta } : : \text{instance}(x) \rightarrow \text{beta } : : \text{seqGet} (\text{seqSingleton}(x), 0) = x) \quad \text{(SeqAxiomC 3)}
\]
\[
\text{seqLen}(\text{seqEmpty}) = 0 \quad \text{(SeqAxiomC 4)}
\]
\[\forall \text{any } b (\text{seqLen}(\text{seqSingleton}(b)) = 1) \quad \text{(SeqAxiomC 5)}
\]
\[
\forall \text{seq } s_1 \forall \text{seq } s_2 (\text{seqLen}(\text{seqConcat}(s_1, s_2)) = \text{seqLen}(s_1) + \text{seqLen}(s_2)) \quad \text{(SeqAxiomC 6)}
\]

\(T\text{h}_{\text{seqC}}^0\) is given by the axioms SeqAxiomC 1 – SeqAxiomC 6.

**Lemma 27**

From \(T\text{h}_{\text{seqC}}^0\) associativity of the seqConcat function is derivable

\[
\forall \text{Seq } s_1, s_2, s_3 (\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)) = \text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3))
\]

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

\[
\forall \text{any } x : : \text{seqGet}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)), i) = \forall \text{any } x : : \text{seqGet}(\text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3), i)
\]

In the following arguments axioms SeqAxiomC 3 and SeqAxiomC 6 are used repeatedly. We distinguish three cases
\(0 \leq i \& i < \text{seqLen}(s_1)\)

\[
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)), i) = \\
\text{any}: \text{seqGet}(s_1, i)
\]

and

\[
\text{any}: \text{seqGet}(\text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3), i) = \\
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, s_2), i) = \\
\text{any}: \text{seqGet}(s_1, i)
\]

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

\[
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)), i) = \\
\text{any}: \text{seqGet}(\text{seqConcat}(s_2, s_3), i - \text{seqLen}(s_1)) = \\
\text{any}: \text{seqGet}(s_2, i - \text{seqLen}(s_1))
\]

and

\[
\text{any}: \text{seqGet}(\text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3), i) = \\
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, s_2), i) = \\
\text{any}: \text{seqGet}(s_2, i - \text{seqLen}(s_1))
\]

\(\text{seqLen}(s_1) + \text{seqLen}(s_2) \leq i \&
\)

\(i < \text{seqLen}(s_1) + \text{seqLen}(s_2) + \text{seqLen}(s_3)\)

\[
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)), i) = \\
\text{any}: \text{seqGet}(\text{seqConcat}(s_2, s_3), i - \text{seqLen}(s_1)) = \\
\text{any}: \text{seqGet}(s_3, i - \text{seqLen}(s_1) - \text{seqLen}(s_2))
\]

and

\[
\text{any}: \text{seqGet}(\text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3), i) = \\
\text{any}: \text{seqGet}(\text{seqConcat}(s_1, s_2), i) = \\
\text{any}: \text{seqGet}(s_3, i - \text{seqLen}(s_1) - \text{seqLen}(s_2))
\]

\[\boxed{}\]

**Lemma 28**

The following two formulas \(\phi_i\) are theorems \(Th_{seqC}^{0}\),
i.e., we can prove \(Th_{seqC}^{0} \vdash \phi_i\).

1. \(\forall s (\text{seqConcat}(s, \text{seqEmpty}) = s)\)

2. \(\forall s (\text{seqConcat}(\text{seqEmpty}, s) = s)\)
**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 $\text{seqEmpty}$, $\text{seqSingleton}$ and $\text{seqConcat}$ is called a constructor term. Thus a constructor term does not contain variables of type $\text{seq}$.

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

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:

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

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

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

Let us first establish a bit of terminology. We call two terms $t$ and $s$ equivalent (in $Th^0_{\text{seqC}}$) if $Th^0_{\text{seqC}} \vdash \forall x(t = n(t))$. 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 $\epsilon$-free if the constant $\text{seqEmpty}$ 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

- $t_0 \equiv \text{seqEmpty}$ or
- $t_0$ is $\epsilon$-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 $\epsilon$-free constructor term

**Definition 21**

*Let $t$ be an $\epsilon$-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 occurrences of the symbol $seqConcat$ in $t$. The number $dct(t)$ is called the defect of $t$. We first observe .18

For all $\epsilon$-free constructor terms $t$: $dct(t) \leq dp(t)$ \hspace{1cm} (2)

The relevant part in the inductive argument is

$$dct(seqConcat(t_0, t_1)) = dct(t_0) + dp(t_1) \hspace{1cm} \text{Def.}$$

$$\leq dp(t_0) + dp(t_1) \hspace{1cm} \text{Ind.Hyp.}$$

$$< dp(t_0) + dp(t_1) + 1 \hspace{1cm} \text{arithmetic}$$

$$= dp(seqConcat(t_0, t_1)) \hspace{1cm} \text{Def.}$$

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

$$dct(seqConcat(t_0, seqSingleton(x))) = dct(seqSingleton(x)) \hspace{1cm} \text{Def 21}$$

$$= dct(t_0) + 0 \hspace{1cm} \text{Def 21}$$

$$= 0 \hspace{1cm} \text{Ind.Hyp}$$

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

Any $\epsilon$-free constructor term $t$ with $dct(t) = 0$ is a normal form. \hspace{1cm} (3)
For the initial cases of the definition of a constructor term this is obvious. For the rest of the argument we investigate an \( \epsilon \)-free constructor term \( t \) of the form \( \text{seqConcat}(t_0, t_1) \) with \( \text{dct}(\text{seqConcat}(t_0, t_1)) = 0 \). Thus \( \text{dct}(t_0) = 0 \) and \( \text{dp}(t_1) = 0 \). The induction hypothesis tells us that \( t_0 \) is a normal form, different from \( \text{seqEmpty} \). Furthermore, we know that \( t_1 \) does not contain the function symbol \( \text{seqConcat} \). Since \( t \) was \( \epsilon \)-free the only possibility is \( t_1 \equiv \text{seqSingleton}(x) \). Our typing restriction exclude terms of the form \( \text{seqSingleton}(\text{seqSingleton}(x)) \). Now, it remains to observe that, if \( t_0 \) is a normal form then also \( \text{seqConcat}(t_0, \text{seqSingleton}(x)) \) is a normal form.

Here comes the next observation:

For all terms \( s_1, s_2, s_3 \)
\[ \text{dct}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3))) > \text{dct}(\text{seqConcat}(s_1, s_2), s_3) \] (4)

This follows from the easy computation:

\[
\begin{align*}
\text{dct}(\text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3))) &= \text{dct}(s_1) + \text{dp}(\text{seqConcat}(s_2, s_3)) \quad \text{Def} \\
&= \text{dct}(s_1) + \text{dp}(s_2) + \text{dp}(s_3) + 1 \quad \text{Def} \\
&> \text{dct}(s_1) + \text{dp}(s_2) + \text{dp}(s_3) \quad \text{arithm.} \\
&= \text{dct}(\text{seqConcat}(s_1, s_2)) + \text{dp}(s_3) \quad \text{Def} \\
&= \text{dct}(\text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3)) \quad \text{Def}
\end{align*}
\]

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

\[ \text{seqConcat}(s_1, \text{seqConcat}(s_2, s_3)) \leadsto \text{seqConcat}(\text{seqConcat}(s_1, s_2), s_3) \]

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 \( \text{dct}(t_f) = 0 \) any thus be a normal form by 3.

Theorem 30

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

\[ \mathcal{M}_{\Sigma_{\text{seq}}}^\mathcal{P} = \mathcal{SQ}_0^0, \]

where \( \mathcal{M}_{\Sigma_{\text{seq}}}^\mathcal{P} \) is then partial structure obtained from \( \mathcal{M} \) (see Definition 9).
Proof Let $\mathcal{M} = (M, I)$. In our restricted setting $M = \mathbb{Z} \cup A \cup \text{Seq}$, with $\mathbb{Z}$, $A$, $\text{Seq}$ the interpretations of the sorts $\text{int}$, $\text{any}$ and $\text{seq}$. For every constructor term $t$ and every assignment $a_1, \ldots, a_n$ of its variables $I(t)(a_1, \ldots, a_n) \in \text{Seq}$. If $t$ is a normal form we write $\langle a_1, \ldots, a_n \rangle$ for $I(t)(a_1, \ldots, a_n) \in \text{Seq}$. 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 $s \in \text{Seq}$ we have $s = \langle a_1, \ldots, a_n \rangle$ for $n = I(\text{seqLen})(s)$ and $a_i = I(\text{any}::\text{seqGet})(s, i)$. Thus $\text{Seq} = \{ \langle a_1, \ldots, a_n \rangle \mid n \in \mathbb{N}, a_i \in A \}$ can be identified with the interpretation of sort $\text{seq}$ 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 $\text{id}$ is an isomorphism, i.e., that the interpretation of the symbols $\text{seqLen}$ and $\text{alpha}::\text{seqGet}$ in $\mathcal{M}$ and $\mathcal{SQ}^0$ agree on the their fixed value domain. Here is a sample. We show that $I(\text{seqLen})(\langle a_1, \ldots, a_n \rangle) = n$. Naturally, this is proved via induction on $n$. The cases $n = 0$ and $n = 1$ are covered by the axioms SeqAxiomC $4$ and SeqAxiomC $5$. Assume that we know $I(\text{seqLen})(\langle a_1, \ldots, a_n \rangle) = n$ and want to convince ourselves of $I(\text{seqLen})(\langle a_1, \ldots, a_n, a_{n+1} \rangle) = n + 1$. Axiom SeqAxiomC $6$ is sufficient to guarantee this. We leave it to the reader to handle the cases $\text{int}::\text{seqGet}$ and $\text{any}::\text{seqGet}$.

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 $\text{int}$ is $\mathbb{Z}$ that goes beyond first order logic.

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

Lemma 31

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

$$\mathcal{M} \models Th^0_{\text{seqC}}$$
Proof Easy.

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

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

Definition 22 ($\text{Th}_{\text{seqC}}^1$)

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

\[
\forall \text{seq } s \forall \text{int } i, j \ (i \leq j \rightarrow \text{seqLen}(\text{seqSub}(s, i, j)) = (j - i) + 1) \tag{SeqAxiomC 7}
\]

\[
\forall \text{seq } s \forall \text{int } i, j, k \ (0 \leq i \land i \leq j \land j < \text{seqLen}(s) \land 0 \leq k \land i + k < j \land \\
\beta :: \text{instance}(\text{seqGet}(s, i + k)) \\
\rightarrow \beta :: \text{seqGet}(\text{seqSub}(s, i, j), k) = \beta :: \text{seqGet}(s, i + k) \tag{SeqAxiomC 8}
\]

In parallel the $\Sigma_{\text{seq}}^0$-structure $\mathcal{S}\mathcal{Q}^0$ is expanded to a $\Sigma_{\text{seq}}^1$-structure $\mathcal{S}\mathcal{Q}^1$

Definition 23

The structure $\mathcal{S}\mathcal{Q}^1 = (U, I)$ coincides with $\mathcal{S}\mathcal{Q}^0$ on the vocabulary $\Sigma_{\text{seq}}^0$ and defines in addition for $s = \langle a_1, \ldots, a_n \rangle$:

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

The analog of Theorem 30 now reads

Theorem 32

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

$\mathcal{M}^p_{\Sigma_{\text{seq}}^1} = \mathcal{S}\mathcal{Q}^1$,

where $\mathcal{M}^p_{\Sigma_{\text{seq}}^1}$ 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) $\Sigma_{seq}^1$-structure that extends $\mathcal{SQ}^1$ then

$$\mathcal{M} \models Th_{seqC}^1$$

Proof Easy.

Definition 24 ($Th_{seqC}^2$)

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

$$\forall seq\ s (seqLen(seqReverse(s)) = seqLen(s)) \quad (SeqAxiomC 9)$$

$$\forall seq\ s \forall int\ k (0 \leq k \land k < seqLen(s) \rightarrow beta ::= seqGet(seqReverse(s),k) = beta ::= seqGet(s,seqLen(s) - k - 1))$$

(SeqAxiomC 10)

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

Definition 25
The structure $\mathcal{SQ}^2 = (U, I)$ coincides with $\mathcal{SQ}^1$ on the vocabulary $\Sigma_{seq}^0$ 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 $\mathcal{M} = (M, I_M)$ be a model of $Th_{seqC}^2$ then

$$\mathcal{M}^{\theta}_{\Sigma_{seq}^2} = \mathcal{SQ}^2,$$
Lemma 35
Let $\mathcal{M}$ be a (total) $\Sigma^2_{\text{seq}}$-structure that extends $\mathcal{SQ}^2$ then

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

Proof Easy.

2.7 An Overspecified Theory of Sequences

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

$$\forall \text{seq } s_1 \forall \text{seq } s_2 (s_1 = s_2 \leftrightarrow \text{seqLen}(s_1) = \text{seqLen}(s_2) \land \forall \text{int } i (0 \leq i \& i < \text{seqLen}(s_1) \rightarrow \text{any} : \text{seqGet}(s_1, i) = \text{any} : \text{seqGet}(s_2, i)))$$

(SeqAxiom 1)

$$\forall \alpha x (\beta :: \text{seqGet}(\text{seqSingleton}(x), 0) = (\beta)x)$$

$$\land \forall \alpha x \forall \text{int } i (i \neq 0 \rightarrow \beta :: \text{seqGet}(\text{seqSingleton}(x), i) = \beta :: \text{seqGet}(\text{seqEmpty}, i))$$

(SeqAxiom 2)

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

$$\forall \text{seq } s_1 \forall \text{seq } s_2 \forall \text{int } i$$

$$i < \text{seqLen}(s_1) \rightarrow$$

$$\beta :: \text{seqGet}(\text{seqConcat}(s_1, s_2), i) = \beta :: \text{seqGet}(s_1, i)$$

$$\land$$

$$\text{seqLen}(s_1) \leq i \rightarrow$$

$$\beta :: \text{seqGet}(\text{seqConcat}(s_1, s_2), i) = \beta :: \text{seqGet}(s_2, i - \text{seqLen}(s_1)))$$

(SeqAxiom 3)
\[
\text{seqLen}(\text{seqEmpty}) = 0 \quad \text{(SeqAxiom 4)}
\]
\[
\forall b (\text{seqLen}(\text{seqSingleton}(b)) = 1) \quad \text{(SeqAxiom 5)}
\]
\[
\forall s_1 \forall s_2 (\text{seqLen}(\text{seqConcat}(s_1, s_2)) = \text{seqLen}(s_1) + \text{seqLen}(s_2)) \quad \text{(SeqAxiom 6)}
\]

SeqAxiom 9 to SeqAxiom 6 make up the theory \( Th^{0}_{\text{seq}} \). 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 \( Th^{0}_{\text{seq}} \) is also a model of \( Th^{0}_{\text{seqC}} \).

**Lemma 36**
There is a model \( M^{0}_{\text{seq}} \) with \( M^{0}_{\text{seq}} \models Th^{0}_{\text{seq}} \)

**Proof by Construction** We assume that only the sorts \text{int}, \text{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 \((\text{int})\), \((\text{any})\) and the predicates \text{int} :: \text{instance}, \text{any} :: \text{instance} have already been defined.

We are now ready for the construction of \( M^{0}_{\text{seq}} = (M, I) \). The universe \( M \) of \( M^{0}_{\text{seq}} \) is given by
\[
M = M_{0} \cup \text{Seq} \\
M_{0} = \mathbb{Z} \cup A
\]

with \( \text{Seq} = \) the set of sequences with element from \( \mathbb{Z} \cup A \). We write elements \( s \in \text{Seq} \) 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 \( \Sigma^{0}_{\text{seq}} \) are interpreted as follows:

\[
\begin{align*}
I(\text{seqEmpty}) & = \langle \rangle \\
I(\text{seqSingleton})(a) & = \langle a \rangle \quad a \in M_{0} \\
I(\text{seqConcat})(\langle s_{0}, \ldots, s_{n-1} \rangle, \langle s_{n}, \ldots, s_{m-1} \rangle) & = \langle s_{0}, \ldots, s_{m-1} \rangle \\
I(\text{seqLen})(\langle s_{0}, \ldots, s_{n-1} \rangle) & = n
\end{align*}
\]

Finally
\[
I(\alpha::\text{seqGet})(\langle s_{0}, \ldots, s_{n-1} \rangle, i) = \begin{cases} 
(\alpha)(s_{i}) & \text{if } 0 \leq i < n \\
0 & \text{otherwise}
\end{cases}
\]

63
An easy inspection shows that indeed SeqAxiom 9 to SeqAxiom 6 are true in the model $M_0$ thus defined. We make use of the assumption that the cast functions and type predicates satisfy their specifications, e.g. $(\text{int})0 = 0$ and $(\text{any})0 = 0$.

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 $\Sigma_0^{seq}$.

**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, $\Sigma_1^{seq} = \Sigma_0^{seq} \cup \{\text{seqSub}\}$ and by adding the following two axioms:

\[
\forall \text{seq } s \forall \text{int } i, j (i \leq j \rightarrow \text{seqLen(seqSub}(s, i, j)) = (j - i) + 1) \quad \text{(SeqAxiom 7)}
\]

\[
\forall \text{seq } s \forall \text{int } i, j (i > j \rightarrow \text{seqLen(seqSub}(s, i, j)) = 0))
\]

\[
\forall \text{seq } s \forall \text{int } i, j, k((0 \leq k \land k \leq j - i \rightarrow \\
\beta :: \text{seqGet(seqSub}(s, i, j), k) = \beta :: \text{seqGet}(s, i + k)) \land \\
k < 0 \lor k < j - i \rightarrow \\
\beta :: \text{seqGet(seqSub}(s, i, j), k) = \beta :: \text{seqGet(seqEmpty}, 0)))
\]

\[
\text{(SeqAxiom 8)}
\]

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

64
Theorem 38

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

$$\mathcal{M}^0_{\Sigma^1_{\text{seq}}} = SQ^1,$$

Proof  Follows from Theorem 32 and the remarks preceding 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., $\text{Th}^1_{\text{seq}} \vdash \forall seq \ s(seqLen(seqSub(s, 1, 0) = 0)$ while this formula is not derivable in $\text{Th}^1_{\text{seqC}}$. Or $\text{Th}^1_{\text{seq}} \vdash \forall seq \ s(any :: seqGet(seqSub(s, 0, 1), 2) = any :: seqGet(s, 2))$ and this formula is also not derivable in $\text{Th}^1_{\text{seqC}}$. 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 $\text{Th}^1_{\text{seq}}$ and $\text{Th}^1_{\text{seqC}}$ agree on total formulas.

Lemma 39

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

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

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

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

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

Obviously SeqAxiom 7 is satisfied.

We now turn to argue that $\mathcal{M}^1_{\text{seq}} \models \text{SeqAxiom 8}$. So let $s = \langle a_0, \ldots, a_{n-1} \rangle$ be an element in $M_{\text{seq}}$ and $i, j, k$ be given with $0 \leq k \land k \leq (j - i)$. If $j < i$ we need to verify
If \( j \geq i \) we get by definition \( I^1(seqSub(s, i, j)) = \langle a_i, \ldots, a_j \rangle \) and thus

\[
\begin{align*}
I^1(beta :: seqGet(seqSub(s, i, j)))(k) &= I^0(beta :: seqGet(seqEmpty))(k) \\
&= 0 \\
&= I^0(beta :: seqGet(seqEmpty))(0) \\
&= I^1(beta :: seqGet(seqEmpty))(0)
\end{align*}
\]

The axioms SeqAxiom 7 and SeqAxiom 8 have been chosen to be as simple as possible neglecting 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 \( \Sigma^1_{seq} \).

**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 choose that axioms simpler than possible. If we had instead of SeqAxiom 8 adopted the even more liberal axiom *SeqAxiom* (*s*)

\[
\forall \text{seq } s\forall \text{int } i, j, k((i \leq j \rightarrow \\
\quad beta :: seqGet(seqSub(s, i, j), k) = beta :: seqGet(s, i + k)) \land \\
\quad i > j \rightarrow \\
\quad beta :: seqGet(seqSub(s, i, j), k) = beta :: seqGet(seqEmpty, 0)))
\]

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
Similarly let $s_1$ denote $(1,1,1,1)$. From $\text{SeqAxiom}(\ast)$ we obtain

$$\text{int} :: \text{seqGet}(\text{seqSub}(s_0,1,0),2) = \text{int} :: \text{seqGet}(s_0,3) = 0$$

and

$$\text{int} :: \text{seqGet}(\text{seqSub}(s_1,1,0),2) = \text{int} :: \text{seqGet}(s_1,3) = 1$$

But, $\text{seqSub}(s_0,1,0) = \text{seqEmpty} = \text{seqSub}(s_1,1,0)$, thus

$$0 = \text{int} :: \text{seqGet}(\text{seqSub}(s_0,1,0),2)
= \text{int} :: \text{seqGet}(\text{seqEmpty},3)
= \text{int} :: \text{seqGet}(\text{seqSub}(s_1,1,0),2)
= 1$$

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

### 2.8 Closing Remarks

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

An extremely valuable instrument of $\text{Th}^{0}_{\text{seq}}$ 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.

$$(\phi[\text{seqEmpty}/s]\land
\forall \text{seq s} \forall \text{any } b(\phi \rightarrow \phi[\text{seqConcat}(s, \text{seqSingleton}(b))/s])) \quad (\text{SeqAxiom } 9)$$

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

$$(\phi[\text{seqEmpty}/s]\land
\forall \text{seq s} \forall \text{alpha } b(\phi \land \text{alpha :: sequence}(s) \rightarrow
\phi[\text{seqConcat}(s, \text{seqSingleton}(b))/s])) \quad (\text{SeqAxiom } 9a)$$

where
\[\text{alpha :: sequence}(s) \equiv s = \text{seqEmpty} \lor \]
\[
\forall \text{int } i((0 \leq i \land i < \text{seqLen}(s)) \rightarrow \\
\text{alpha :: instance(any::seqGet}(s,i))\]

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 central 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

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

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

69
Definition 26 (Conservative Extension)

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

$$T_0 \vdash \phi \iff T_1 \vdash \phi$$

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 $T_{BF}^0 = \{ (1.1), (1.2), (1.3), (1.4) \& (1.5), (1.6), (1.7), (1.8) \}$ and $T_{BF}^1 = T_{BF}^0 \cup \{(1.11)\}$. Is $T_{BF}^1$ a conservative extension of $T_{BF}^0$? 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 $M_0$ be a $\Sigma_0$-structure and $M_1$ an $\Sigma_1$-expansion of $M_0$. Then

$$M_0 \models \phi \iff M_1 \models \phi$$

Proof Obvious.

Definition 27 (Expansion)

Let $\Sigma_0 \subseteq \Sigma_1$ be signatures.

1. A $\Sigma_1$-structure $M_1 = (M_1, I_1)$ is called an expansion of a $\Sigma_0$-structure $M_0 = (M_0, I_0)$ 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 \(\Sigma_1\)-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 \(\Sigma_0\) 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 \(Fml_{\Sigma_i}\).

\(T_1\) is called a semantic conservative extension of \(T_0\) if

1. for all \(\Sigma_1\)-structures \(M_1\)
   \[M_1 \models T_1 \Rightarrow (M_1 \upharpoonright \Sigma_0) \models T_0\]

2. for every \(\Sigma_0\)-structure \(M_0\) with \(M_0 \models T_0\) there is a \(\Sigma_1\)-expansion \(M_1\) of \(M_0\) with \(M_1 \models T_1\).

**Theorem 42 (Criterion for Conservative Extension)**

Let \(\Sigma_0 \subseteq \Sigma_1\) be signatures, and \(T_i\) sets of sentences in \(Fml_{\Sigma_i}\).

If \(T_1\) is a semantic conservative extension of \(T_0\) then it is also a (syntactic) conservative extension.

**Proof** Let \(\phi\) be a sentence in \(Fml_{\Sigma_0}\) with \(T_0 \vdash \phi\). Let \(M_1\) be an arbitrary \(\Sigma_1\)-structure. By assumption \((M_1 \upharpoonright \Sigma_0) \models T_0\). Thus we also have \((M_1 \upharpoonright \Sigma_0) \models \phi\). By the coincidence lemma 41 we also have \(M_1 \models \phi\). In total we have shown \(T_1 \vdash \phi\).

Now, assume \(T_1 \vdash \phi\). If \(M_0\) is an arbitrary \(\Sigma_0\)-structure there is by the assumption an expansion of \(M_0\) to a \(\Sigma_1\)-structure \(M_1\). From \(T_1 \vdash \phi\) we thus get \(M_1 \models \phi\). The coincidence lemma 41 tells us again that also \(M_0 \models \phi\). In total we arrive at \(T_0 \vdash \phi\).

**Lemma 43 (Definitional Extension)**

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

Let \(t\) be a term in \(\Sigma_0\) 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 *definition*al 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 $T_{1BF}$ is a definitional extension of $T_{0BF}$.

**Lemma 44 (Definable Extension)**

Let $T_0$ be a set of sentences in the signature $\Sigma_0$. Let $\Sigma_1 = \Sigma_0 \cup \{f\}$ for a new $n$-place function symbol $f$ and let $\phi$ be a $\Sigma_0$ 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 $\psi$ be a formula with free variables $w_0, w_1, \ldots, w_n$ in the signature $\Sigma$ and $T$ a $\Sigma$-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 $\Sigma$ (sometimes denoted in greater detail as $f_{\exists w_0 \psi}$) and

2. adding the axiom

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

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 $\psi$ be the formula addressed in Lemma 45 with free variables $w, w_1, \ldots, w_n$. Use Lemma 44 for the formula

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

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 separate 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 $\Sigma$-model of $T$. We define for $a_1, \ldots, a_n = \bar{a} \in M$

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

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

---

**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 $\Sigma_{DG}$ consists of one binary relation $N(x, y)$. If $\mathcal{G} = (G, I)$ is a $\Sigma_{DG}$ 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

\[
\begin{align*}
\forall x (\neg x < x) & \quad \text{strictness} \\
\forall x, y, z (x < y \land y < z \rightarrow x < z) & \quad \text{transitivity} \\
\forall x, y (N(x, y) \rightarrow x < y) & \quad \text{compatibility}
\end{align*}
\]

Thus $<$ is a strict order relation that is compatible with the edge relation.
If $G = (G, I)$ is a model of $T_{ODG}$ the relation $I(<)$ is frequently called a topological ordering 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 $T_{ODG} \vdash \neg \exists x, y, z (N(x, y) \land N(y, z) \land N(y, z))$

but $T_{DG} \nvdash \forall x \neg N(x, x)$ and $T_{DG} \nvdash \neg \exists x, y, z (N(x, y) \land N(y, z) \land N(y, z))$

### 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 $Diag(M)$ of a structure $M$.

**Definition 29**

Let $M$ be a $\Sigma$-structure. The signature $\Sigma_M$ is obtained from $\Sigma$ by adding new constant symbols $c_a$ for every element $a \in M$.

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

**Definition 30 (Diagram of a structure)**

Let $M$ be a $\Sigma$-structure. The diagram of $M$, in symbols $Diag(M)$, is defined by

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

**Lemma 47**

Let $M$ be a $\Sigma$-structure.

If $N \models Diag(M)$ then $M$ is (isomorphic to) a substructure of $N$.

**Proof** Easy.

Furthermore, we will need the following observation:

**Lemma 48**

Let $M_0$ be a substructure of $M$ and $\phi$ logically equivalent to a universal sentence. Then

$$M \models \phi \Rightarrow M_0 \models \phi$$
Proof Easy induction on the complexity of $\phi$.

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 $\Sigma$-structures. $\mathcal{M}_0$ is called a substructure of $\mathcal{M}$ iff

1. $M_0 \subseteq M$

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 $\text{Fml}_{\Sigma_i}$ 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 $\Sigma_1$-structure with $\mathcal{M}_1 \models T_1$ and $\mathcal{M}_0$ its restriction to $\Sigma_0$, i.e., $\mathcal{M}_0 = \mathcal{M}_1 \upharpoonright \Sigma_0$. 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 $\Sigma_0$-structure $\mathcal{M}_0$ with $\mathcal{M}_0 \models T_0$. We set out
to find an expansion $M_1$ of $M_0$ with $M_1 \models T_1$. To this end we consider the theory $T_1 \cup \text{Diag}(M_0)$. If this theory were inconsistent then already $T_1 \cup F$ for a finite subset $F \subseteq \text{Diag}(M_0)$ would be inconsistent. This is the same as saying $T_1 \cup F$ for a finite subset $F \subseteq \text{Diag}(M_0)$ would be inconsistent. This is the same as saying $T_1 \models \neg F$. Since the constants $c_a$ do not occur in $T_1$ we get furthermore $T_1 \models \forall x_1, \ldots, x_n \neg F'$, where $F'$ is obtained from $F$ by replacing all occurrences of constants $c_a$ by the same variable $x_i$. This is equivalent to $T_1 \models \neg \exists x_1, \ldots, x_n F'$. Since $T_1$ was assumed to be a conservative extension of $T_0$ we also get $T_0 \models \neg \exists x_1, \ldots, x_n F'$ and thus $M_0 \models \neg \exists x_1, \ldots, x_n F'$. This is a contradiction since by the definition of $\text{Diag}(M_0)$ we have $M_0 \models \exists x_1, \ldots, x_n F'$ by instantiating the quantified variable $x_i$ that replaces the constant $c_a$ by the element $a$. This contradiction shows that $T_1 \cup \text{Diag}(M_0)$ is consistent. Let $\mathcal{N}$ be a model of this theory. By Lemma 47 we may assume that $M_0$ is a substructure of $(\mathcal{N} \upharpoonright \Sigma_0)$. Since by assumption only new relation symbols are added when passing from $\Sigma_0$ to $\Sigma_1$ 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 $(\mathcal{N} \upharpoonright \Sigma_0) = M_0$ 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 $\Gamma$ 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 $\omega$

Theory $T_0$ is given by the axioms

1. $\forall x R(x, f(x))$
2. $\forall x \forall y \forall z (R(x, y) \land R(y, z) \rightarrow R(x, z))$
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 $\Sigma_0$-formula $\phi$.

Now assume $T_1 \vdash \phi$. We want to show $T_0 \not\vdash \phi$. Assume this is not the case. Thus there is a $\Sigma_0$-model $M$ of $T_0 \cup \{\neg \phi\}$. The axioms of $T_0$ imply that for all $n \in \mathbb{N}$ we have $M \models R(f^n(0), f^{n+1}(0))$ and the elements $0^M$, $f(0)^M$, $\ldots f^n(0)^M$ are all different. In particular $M \models R(f^i(0), f^n(0))$ 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 $N$. If we define $B^N = \{f^i(0)^N \mid i \in \mathbb{N}\}$ then $N_1$ satisfies axiom 4. and 5. of $T_1$. If $N \models \omega = f^n(0)$ we would have $N \models R(\omega, \omega)$ which contradicts axiom 3 of $T_0$. This $N_1$ is a model of $T_1 \cup \{\neg \phi\}$ contradicting our assumption. This completes the proof by contradiction of $T_0 \not\vdash \phi$.

$T_1$ is not a conservative semantic extension of $T_0$

Let $M$ by a structure with universe $M = \mathbb{N}$, $0^M = 0$, $f^M(n) = n + 1$ and $R^M(n, m) \iff n < m$. It can be easily checked that $M \models T_0$. If $M_1$ were an $\Sigma_1$-expansion of $M$ with $M_1 \models T_1$ axiom 5 entails $B^{M_1} = \mathbb{N} = M$. 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 $fwPath$

$fwPath$

\begin{verbatim}
\begin{schemaVar}
  \term Seq seq;
\end{schemaVar}

\begin{schemaVar}
  \variables int iv;
\end{schemaVar}

\find(fwpath(seq))

\begin{varcond}(\notFreeIn(iv, seq))
\end{varcond}

\begin{replacewith}(\forall iv;((0 \leq iv \land iv \lt seqLen(seq) - 1) \rightarrow
    edge(node::seqGet(seq, iv), node::seqGet(seq, iv + 1))))
\end{replacewith}
\end{verbatim}

Taclet Forward Structural Seq Induction

$seqInd_{\text{forward}}$

\begin{verbatim}
\begin{varcond}(\notFreeIn(m, b))
\end{varcond}

"Base Case": \add(\Longrightarrow \{\text{subst } s; \text{seqEmpty}\}(b));

"Step Case": \add(\Longrightarrow \forall s;\{}\text{forall } m;
    (b\rightarrow\{\text{subst } s; \text{seqConcat}(s, \text{seqSingleton}(m))\}b) \})

"Use Case": \add(\forall s;(b \Longrightarrow)
\end{verbatim}

Taclet Backward Structural Seq Induction
seqInd.backward{
  varcond (\ notFreeIn (m,b))
  "Base Case": \add ( \implies \{ \subst s; \seqEmpty\}(b) );
  "Step Case": \add ( \implies \forall s; (\forall m; 
    (b\implies\{ \subst s; (\seqConcat (\seqSingleton (m), s))\} b) ));
  "Use Case": \add ( \forall s; (b \implies)
};

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 \textit{fdpath}

fdPath{ \schemaVar \term Seq seq; 
  \schemaVar \variables int iv;
  \find (fdpath(seq))
  \varcond (\ notFreeIn (iv, seq))
  \replacewith (fwpath(seq) \& 
    \forall iv; ((0 \leq iv \& iv < \seqLen (seq)−1) \implies 
      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;
}
\textbf{functions} { \\
// getters 
alpha \alpha::\text{seqGet}(\text{Seq}, \text{int})
int \text{seqLen}(\text{Seq}); \\

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

\textbf{rules} { \\
// \text{getOfSeqSingleton} { 
\text{schemaVar} \text{\term alpha} x; 
\text{schemaVar} \text{\term int} idx; 
\text{find}(\beta::\text{seqGet}(\text{seqSingleton}(x), \text{idx})) 
\text{replacewith}((\text{if} (\text{idx} = 0)) 
\text{then}((\beta)x) 
\text{else}(\beta::\text{seqGet}(\text{seqEmpty}, 0))) 
\text{heuristics(simplify)} 
}; 

getOfSeqConcat { 
\text{schemaVar} \text{\term Seq} seq, seq2; 
\text{schemaVar} \text{\term int} idx; 
\text{find}(\beta::\text{seqGet}(\text{seqConcat}(seq, seq2), \text{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;

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

  heuristics(simplify_enlarging)
}

81
\( \text{find(seqLen(seq))} \)

\( \text{add}(0 \leq \text{seqLen(seq)} \Rightarrow) \)

\( \text{heuristics(inReachableStateImplication)} \)

};

lenOfSeqEmpty { 
\( \text{find(seqLen(seqEmpty))} \)

\( \text{replacewith}(0) \)

\( \text{heuristics(concrete)} \)

};

lenOfSeqSingleton {
\( \text{schemaVar \ term alpha x;} \)

\( \text{find(seqLen(seqSingleton(x)))} \)

\( \text{replacewith}(1) \)

\( \text{heuristics(concrete)} \)

};

lenOfSeqConcat {
\( \text{schemaVar \ term Seq seq, seq2;} \)

\( \text{find(seqLen(seqConcat(seq, seq2)))} \)

\( \text{replacewith(seqLen(seq) + seqLen(seq2))} \)

\( \text{heuristics(simplify)} \)

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

\text{find}(\text{seqLen}(\text{seqSub}(seq, from, to)))

\text{replacewith}(\ 
  \text{if}(from \leq to)\text{then}(((to - from) + 1)\text{else}(0))

\text{heuristics}(\text{simplify_enlarging})
};

lenOfSeqReverse { 
  \text{schemaVar} \text{ term Seq seq;}

\text{find}(\text{seqLen}(\text{seqReverse}(seq)))

\text{replacewith}(\text{seqLen}(seq))

\text{heuristics}(\text{simplify})
};

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

\text{find}(s = s2)
  \text{varcond}(\neg \text{FreeIn}(iv, s, s2))

\text{replacewith}(\text{seqLen}(s) = \text{seqLen}(s2)
  & \forall \text{forall} \text{ iv; } (0 \leq \text{iv} \& \text{iv} < \text{seqLen}(s)
  \rightarrow
  \text{any::seqGet}(s, \text{iv}) = \text{any::seqGet}(s2, \text{iv})))
};

equalityToSeqGetAndSeqLenLeft { 
  \text{schemaVar} \text{ term Seq s, s2;}

83
schemaVar variables int iv;

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

\add(seqLen(s) = seqLen(s2)
& \forall iv; (0 <= iv & iv < seqLen(s)

\rightarrow
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)

\rightarrow
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;
\texttt{seqSingleton} (x) = \texttt{EQ} \implies \texttt{seqGet} (\texttt{EQ}, \texttt{idx})
\texttt{replacewith} (\texttt{if} (\texttt{idx} = 0)
\texttt{then} ((\texttt{beta})x)
\texttt{else} (\texttt{seqGet} (\texttt{seqEmpty}, 0)))
\texttt{heuristics} (\texttt{simplify})
}\;
\texttt{getOfSeqConcatEQ} \{\texttt{seqVar} \texttt{term} \texttt{Seq} \texttt{seq, seq2}; \texttt{seqVar} \texttt{term} \texttt{int} \texttt{idx}; \texttt{seqVar} \texttt{term} \texttt{Seq} \texttt{EQ};\texttt{assumes} (\texttt{seqConcat} (\texttt{seq}, \texttt{seq2}) = \texttt{EQ} \implies \texttt{seqGet} (\texttt{EQ}, \texttt{idx}))
\texttt{replacewith} (\texttt{if} (\texttt{idx} < \texttt{seqLen} (\texttt{seq}))
\texttt{then} (\texttt{seqGet} (\texttt{seq}, \texttt{idx}))
\texttt{else} (\texttt{seqGet} (\texttt{seq2}, \texttt{idx} \texttt{-} \texttt{seqLen} (\texttt{seq}))))
\texttt{heuristics} (\texttt{simplify_enlarging})
}\;
\texttt{getOfSeqSubEQ} \{\texttt{seqVar} \texttt{term} \texttt{Seq} \texttt{seq}; \texttt{seqVar} \texttt{term} \texttt{int} \texttt{idx}, \texttt{from}, \texttt{to}; \texttt{seqVar} \texttt{term} \texttt{Seq} \texttt{EQ};\texttt{assumes} (\texttt{seqSub} (\texttt{seq}, \texttt{from}, \texttt{to}) = \texttt{EQ} \implies \texttt{seqGet} (\texttt{EQ}, \texttt{idx}))
\texttt{replacewith} (\texttt{seqGet} (\texttt{seq}, \texttt{idx} + \texttt{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
\[ \text{replacewith}(1) \]
\[ \text{heuristics}(\text{concrete}) \]
\[ \text{lenOfSeqConcatEQ} \{ \]
\[ \text{schemaVar} \ \text{term Seq seq, seq2} ; \]
\[ \text{schemaVar} \ \text{term Seq EQ} ; \]
\[ \text{assumes} (\text{seqConcat}(seq, seq2) = \text{EQ} \implies) \]
\[ \text{find}(\text{seqLen}(\text{EQ})) \]
\[ \text{sameUpdateLevel} \]
\[ \text{replacewith}(\text{seqLen}(seq) + \text{seqLen}(seq2)) \]
\[ \text{heuristics}(\text{simplify}) \]
\[ \} ; \]
\[ \text{lenOfSeqSubEQ} \{ \]
\[ \text{schemaVar} \ \text{term Seq seq} ; \]
\[ \text{schemaVar} \ \text{term int from, to} ; \]
\[ \text{schemaVar} \ \text{term Seq EQ} ; \]
\[ \text{assumes} (\text{seqSub}(seq, from, to) = \text{EQ} \implies) \]
\[ \text{find}(\text{seqLen}(\text{EQ})) \]
\[ \text{sameUpdateLevel} \]
\[ \text{replacewith}( \text{if}(\text{from} \leq \text{to}) \text{then}((\text{to} - \text{from}) + 1) \text{else} 0) \]
\[ \text{heuristics}(\text{simplify_enlarging}) \]
\[ \} ; \]
\[ \text{lenOfSeqReverseEQ} \{ \]
\[ \text{schemaVar} \ \text{term Seq seq} ; \]
\[ \text{schemaVar} \ \text{term Seq EQ} ; \]

87
\assumes (\text{seqReverse}(\text{seq}) = \text{EQ} \implies) \\
\text{find}(\text{seqLen}(\text{EQ})) \\
\text{sameUpdateLevel} \\
\replacewith (\text{seqLen}(\text{seq})) \\
\heuristics (\text{simplify}) \\
}; \\
//-------------------------------------------------- \\
//lemmata for seqEmpty \\
//-------------------------------------------------- \\
\text{concatWithEmpty1} \{ \\
\text{schemaVar} \ \text{term} \ \text{Seq} \ \text{seq}; \\
\text{find}(\text{seqConcat}(\text{seq}, \text{seqEmpty})) \\
\replacewith(\text{seq}) \\
\heuristics (\text{concrete}) \\
\}; \\
\text{concatWithEmpty2} \{ \\
\text{schemaVar} \ \text{term} \ \text{Seq} \ \text{seq}; \\
\text{find}(\text{seqConcat}(\text{seqEmpty}, \text{seq})) \\
\replacewith(\text{seq}) \\
\heuristics (\text{concrete}) \\
\}; \\
//-------------------------------------------------- \\
//lemma for casts \\
//-------------------------------------------------- \\
\text{castedGetAny} \{ \\
\text{schemaVar} \ \text{term} \ \text{Seq} \ \text{seq}; \\
}
\texttt{\textbackslash{}schemaVar} \texttt{\textbackslash{}term int idx;}
\texttt{\textbackslash{}find ((beta)any::seqGet(seq, idx))}
\texttt{\textbackslash{}replacewith (beta::seqGet(seq, idx))}
\texttt{\textbackslash{}heuristics (simplify)}
\}
\}

\section{4.3 Taclets for bSum}

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

\texttt{\textbackslash{}rules} {
//
// rules for bounded sums
//

\texttt{bsum\_split} {
\texttt{\textbackslash{}find (bsum\{uSub;\} (i0, i2, t))}
\texttt{\textbackslash{}varcond ( \notFreeIn(uSub1, i0),}
\texttt{\notFreeIn(uSub1, i1),}
\texttt{\notFreeIn(uSub1, i2),}
\texttt{\notFreeIn(uSub, i0),}
\texttt{\notFreeIn(uSub, i1),}
\texttt{\notFreeIn(uSub, i2),}
\texttt{\notFreeIn(uSub1, t) )}
\texttt{\textbackslash{}replacewith (\textbackslash{}if(i0\leq i1 & i1\leq i2)
\texttt{\textbackslash{}then(bsum\{uSub;\}(i0, i1, t) +
\texttt{bsum\{uSub1;\}(i1,i2,\{\textbackslash{}subst uSub; uSub1\}t))}
\texttt{\textbackslash{}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),

\texttt{notFreeIn(uSub, i2))}
\texttt{replacewith(bsum\{uSub:} (i0 \hspace{1mm} 1, i2, t) – }
\texttt{if(i0 \hspace{1mm} 1 < i2)}
\texttt{then(\texttt{\{subst uSub; (INT2)(i0 \hspace{1mm} 1})t) else (0))}

\texttt{bsum\_induction\_lower2\_concrete \{}
\texttt{find(bsum\{uSub:) (1+i0, i2, t))}
\texttt{varcond (\texttt{notFreeIn(uSub, i0),} }
\texttt{notFreeIn(uSub, i2))}
\texttt{replacewith(bsum\{uSub:) (i0, i2, t) – }
\texttt{if(i0 < i2)}
\texttt{then(\texttt{\{subst uSub; (INT2)(i0})t) else (0))}
\texttt{heuristics(simplify)}
\texttt{\}};

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

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

\texttt{bsum\_equal\_split1 \{}
\texttt{find(==> bsum\{uSub1:) (i0, i1, t1)}
\texttt{= bsum\{uSub2:) (i0, i2, t2))}
\texttt{varcond (\texttt{notFreeIn(uSub1, i0),} }
\texttt{notFreeIn(uSub2, i2))}
\texttt{\}};

92
\notFreeIn(uSub1, i1),
\notFreeIn(uSub1, i2),
\notFreeIn(uSub1, t2),
\notFreeIn(uSub2, i2),
\notFreeIn(uSub2, i1),
\notFreeIn(uSub2, t1),
\notFreeIn(uSub2, t0))
\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, i1),
   \notFreeIn(uSub2, t1),
   \notFreeIn(uSub2, i0)
)
\add(==) i1<=i0 & i2<=i0 &
   \if (i2<i1)
      \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 , 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_concretel { 
  \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(negl(jz)), t))  
\varcond(\notFreeIn(uSub, iz),  
  \notFreeIn(uSub, jz))  
\replacewith(  
  \if(Z(negl(jz)) \leq Z(iz))  
  \then(0)  
  \else(bsum\{uSub;\}(Z(iz), Z(negl(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 \implies)  
\heuristics(simplify)  
};

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

\[ \text{if}(b) \text{then}(0) \text{else}(1)) \text{ sameUpdateLevel} \]
\[ \text{varcond} \ (\not\text{FreeIn}(uSub, i0), \not\text{FreeIn}(uSub, i1)) \]
\[ \text{add}(\text{bsum}\{uSub:\} (i0, i1, \text{if}(b) \text{then}(0) \text{else}(1)) \geq 0 \implies \]
\[ \text{heuristics} (\text{simplify}) \]

\text{equal\_bsum1} \{}
\[ \text{find}(\implies \ bsum\{uSub1:\} (i0, i1, t1) = \]
\[ bsum\{uSub2:\} (i0, i1, t2)) \]
\[ \text{varcond} \ (\not\text{FreeIn}(uSub2, t1), \not\text{FreeIn}(uSub1, t2), \not\text{FreeIn}(uSub1, i0), \not\text{FreeIn}(uSub1, i1), \not\text{FreeIn}(uSub2, i0), \not\text{FreeIn}(uSub2, i1)) \]
\[ \text{add}(\implies \forall uSub1; ((uSub1 \geq i0 \& uSub1 < i1) \rightarrow \]
\[ t1=\{ subst uSub2; uSub1\} t2)) \]
\[ \text{heuristics} (\text{comprehensions\_high\_costs}) \]
\}

\text{equal\_bsum2} \{}
\[ \text{assumes}(bsum\{uSub1:\} (i0, i1, t1) = i \implies) \]
\[ \text{find}(\implies \ bsum\{uSub2:\} (i0, i1, t2) = i) \]
\[ \text{varcond} \ (\not\text{FreeIn}(uSub2, t1), \not\text{FreeIn}(uSub1, t2), \not\text{FreeIn}(uSub1, i0), \not\text{FreeIn}(uSub1, i1), \not\text{FreeIn}(uSub2, i0), \not\text{FreeIn}(uSub2, i1)) \]
\[ \text{add}(\implies \forall uSub1; ((uSub1 \geq i0 \& uSub1 < i1) \rightarrow \]
\[ t1=\{ subst uSub2; uSub1\} t2)) \]
\[ \text{heuristics} (\text{comprehensions\_high\_costs}) \]
\}

\text{equal\_bsum3} \{}
\[ \text{assumes}(bsum\{uSub1:\} (i0, i1, t1) = i, \]
\[ bsum\{uSub2:\} (i0, i1, t2) = j \implies) \]
\[ \text{find}(\implies \ j = i) \]

\}
\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) -> 
\quad t1=(\{\sub uSub2; uSub1\}t2))) 
\heuristics(\comprehensions_high_costs) 
};

equal_bsum_zero_cut { 
\find(=>> bsum\{uSub1;\} (i0, i1, t1) = 
\quad bsum\{uSub2;\} (i2, i3, t2)*t) 
\add( \implies bsum\{uSub1;\} (i0, i1, t1)=0); 
\add( bsum\{uSub1;\} (i0, i1, t1)=0 \implies) 
\heuristics(\comprehensions_high_costs) 
};
pullOutbsum1 { 
\find(bsum\{uSub1;\} (i0, i1, t1) >= t =>> 
\quad \varcond ( \new(sk, \dependingOn(t1)),2 
\quad \new(sk, \dependingOn(i0)), 
\quad \new(sk, \dependingOn(i1)) ) 
\replaceWith (sk >= t ==>) 
\add ( bsum\{uSub1;\} (i0, i1, t1) = sk =>> 
\heuristics(simplify) 
};
pullOutbsum2 { 
\find(bsum\{uSub1;\} (i0, i1, t1) <= t =>> 
\quad \varcond ( \new(sk, \dependingOn(t1)), 
\quad \new(sk, \dependingOn(i0)), 
\quad \new(sk, \dependingOn(i1)) ) 
\replaceWith (sk <= t ==>) 
\add ( bsum\{uSub1;\} (i0, i1, t1) = sk =>> 
\heuristics(simplify) 
};
}
Bibliography


