Universität Ulm

Fakultät für Informatik

From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

Klaus Achatz, Helmuth Partsch

Universität Ulm

Nr. 96-13

Ulmer Informatik-Berichte

Dezember 1996

From Descriptive Specifications to Operational ones:

A Powerful Transformation Rule, its Applications and Variants

K. Achatz, H. Partsch
Fakultät für Informatik
Universität Ulm

Abstract

This paper introduces, discusses and proves a transformation rule to convert specifications of set-valued functions defined by set comprehension into functional implementations. The power of the rule is illustrated by several examples, among them a Prolog interpreter. Also, variants of the rule for specifications involving existential quantification and arbitrary choice are presented and illustrated by representative examples.

0. Introduction

Transformational programming is short for a methodology of constructing programs from formal problem specifications by stepwise application of formal, semantics-preserving transformation rules. In the following the reader is assumed to be basically familiar with the idea of transformational programming and its basic principles. A brief introduction and an overview can be found, e.g., in [Bauer et al. 89] or [Boiten et al. 92]. For a comprehensive treatment of the subject, cf., e.g., [Partsch 90].

In this paper we mainly introduce a powerful transformation rule to convert specifications of set-valued functions defined by set comprehension into executable functional implementations. This rule can be profitably used for all kinds of problems that ask for a set of complex objects that may somehow be built up incrementally, possibly involving backtracking. The rule emerged when dealing with a formal derivation of a Prolog interpreter from a descriptive specification. It turned out that most transformation steps in this concrete exercise were independent of the particular problem and that the remaining ones could be abstracted appropriately. Meanwhile, the rule has been successfully applied to numerous non-trivial examples, some of which are presented in this paper. ~~More examples and a detailed treatment of those presented here can be found in [Achatz, Partsch 96].~~

0.1 Notational conventions

Within our rules and examples, we use the following notational conventions.

0.1.1 Fonts

Italic is used for all bound variables. Scheme parameters (for expressions in rules) are denoted by upright-uppercase letters (augmented with "arguments" to indicate potential dependencies). And for types (and type variables) we employ boldface-lowercase symbols.

0.1.2 Data types

Type variables (in program schemes) are denoted by single boldface letters (such as **m** or **n**). As basic types, we assume to have available natural numbers (with type **nat** and the usual operations) as well as booleans.

Based on basic types, more complex ones may be built by forming supertypes (e.g., **m** | **n** comprising all elements of types **m** and **n**) and record types (e.g. (*a*: **m**, *b*: **n**) denoting pairs of elements of types **m** and **n** respectively, where *a* and *b* are selectors for the first (second) argument).

More complex types we will use are sets, sequences, and maps. For respective formal definitions in terms of algebraic types we refer to [Partsch 90]. The intuition of the basic operations of these types (appearing in the examples) is as follows:

sets (type **set of**) with operations

- $\emptyset$ the empty set
- $\cup$ set union
- $\subseteq$ subset relation
- $\in$ element relation

sequences (type **sequ of**) with operations

- [] the empty sequence
- **hd** (**lst**) first (last) element of a (non-empty) sequence
- **tl** (**fst**) (non-empty) sequence without first (last) element
- ++ concatenation of sequences (also used to append single elements)
- $s - x$ removal of element x from sequence s
- $s[i]$ i-th element of a (non-empty) sequence s
- $|s|$ length of sequence s
- $\in$ element relation

maps (correspondences defined by an algebraic type EMAP, cf. [Partsch 90]) with operations

- $\emptyset$ the empty map
- $\circ$ composition of maps

- $m[i]$ the "value" of map m for "argument" i
- *dom* the domain of the map
- *ran* the range of the map
- $m[i] \leftarrow a$ map m with "value" of i-th "argument" updated to a.

0.1.3 Programs and program schemes

For programs and program schemes a mainly self-explanatory Pascal-like notation is used with a strict, call-by-value semantics. For a complete definition of the language used, we refer to [Bauer et al. 85]. Some of the less conventional notations (and their meaning) are:

$\{a$: **m** $\Vert$ $\mathrm{P}(a)\}$	set comprehension, denoting the set of all elements a (of type **m**) for which $\mathrm{P}(a)$ holds
some a: **m** $\Vert$ $\mathrm{P}(a)$	(non-deterministic) choice, denoting some element a (of type **m**) which satisfies $\mathrm{P}(a)$
$(a$: **m** $\Vert$ $\mathrm{P}(a))$	subtype of **m**, comprising all elements a (of type **m**) which fulfil $\mathrm{P}(a)$; also used for assertions in the domain of functions
$\vert\{\ldots\}\vert$	cardinality of a set
Δ (∇)	sequential conjunction (disjunction)

0.1.4 Transformation rules

Transformation rules are denoted as $\begin{array}{c} I \\ \updownarrow \\ O \end{array} \big| C$ resp. $\begin{array}{c} I \\ \downarrow \\ O \end{array} \big| C$.

I is the input scheme of the rule, O its output scheme. The double arrow means semantic equivalence, the single arrow descendance. C is a set of applicability conditions, all of which are (implicitly) universally quantified clauses (consisting of a premise, $\vdash$ as an implicational symbol, and a semantic relation – equivalence or descendance – as a consequence). Within applicability conditions, $\equiv$ is used to denote semantic equivalence, and $\subseteq$ to denote descendance. If there is no premise, also $\vdash$ will be left out. The syntactic constraints (following a rule) give sufficient (but not necessarily minimal) information to infer complete typing of all expressions and functions involved. For details on the logical background of transformation rules, cf. [Partsch 90].

In order to shorten the presentation of the rules we furthermore use the following

General convention

In all transformation rules definedness and determinacy are assumed for all expression symbols and auxiliary functions. Also, all program schemes are supposed to be syntactically valid and context-correct.

0.2 Outline of the following sections

Section 1 introduces our central transformation rule, gives some intuition on how to use it, and provides its correctness proof. Section 2 deals with some applications of the rule. In this context, the N-Queens problem will be dealt with in all details (including all instantiations and proof obligations). The problem of all segmentations of a sequence and a Prolog interpreter are treated in a condensed form. In section 3 some variants of the rule from section 1 are discussed. These variants deal with other specification constructs such as existential quantification or non-deterministic choice and are also illustrated by examples (from the area of parsing). As a particular variant we will also show in this section that, under additional constraints, our rule from section 1 indeed simplifies to a form one would intuitively expect. Section 4, finally, deals with related work and draws some conclusions.

1. A powerful transformation rule

In this section, we present our transformation rule, give some intuition and comments, and prove the correctness of the rule.

1.1 The rule

$f(X)$ **where**

$f(x) =_{\text{def}} \{r(X, y)\text{: } \mathbf{n} \parallel \mathrm{Q}(x, y)\}$

(0) $(|\{r(X, y) \parallel \mathrm{Q}(x, y)\}| < \infty) \equiv \mathbf{true}$

(1) $\mathrm{P}(X, X, \mathrm{E}) \equiv \mathbf{true}$

(2) $\mathrm{Q}(x, y) \equiv \mathrm{Q'}(x, \mathrm{E}, y)$

(3) $\mathrm{P}(X, u, v) \;\Delta\; \mathrm{T}(u, v) \equiv \mathbf{true} \vdash \mathrm{H}(u, v) \equiv \{r(X, y) \parallel \mathrm{Q'}(u, v, y)\}$

(4) $\mathrm{P}(X, u, v) \;\Delta\; \neg\mathrm{T}(u, v) \equiv \mathbf{true} \vdash$

$\{r(X, y) \parallel \mathrm{Q'}(u, v, y)\} \equiv$

$\mathrm{D}(u, v) \cup \bigcup_{i = 1..n(u, v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \;\Delta\; \mathrm{Q'}(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

(5) $1 \le i \le n(u, v) \;\Delta\; \mathrm{P}(X, u, v) \;\Delta\; \neg\mathrm{T}(u, v) \;\Delta\; \mathrm{B}(i, u, v) \equiv \mathbf{true} \vdash$

$\mathrm{P}(X, \mathrm{K}(i, u, v), \mathrm{R}(i, u, v)) \equiv \mathbf{true}$

(6) $1 \le i \le n(u, v) \;\Delta\; \mathrm{B}(i, u, v) \equiv \mathbf{true} \vdash$

$(\mathrm{R}(i, u, v) < v) \equiv \mathbf{true}$ **where** WF-ORD(**r**, <) **or**

$(\mathrm{K}(i, u, v) < u) \equiv \mathbf{true}$ **where** WF-ORD(**m**, <)

(7) $next(k, u, v) \subseteq$

some k': **nat** $\parallel k' \in \{i\text{: } \mathbf{nat} \parallel k{+}1 \le i \le n(u, v){+}1 \;\Delta$

$\forall\, i'$: **nat** $\parallel k < i' < i \Rightarrow \neg\mathrm{B}(i', u, v)\}$

$g(X)$ **where**

$g(x) =_{\mathrm{def}} g'(x, \mathrm{E}, \emptyset)$

$g'(u, v, s) =_{\mathrm{def}}$ **if** $\mathrm{T}(u, v)$ **then** $s \cup \mathrm{H}(u, v)$ **else** $g''(next(0, u, v), u, v, s \cup \mathrm{D}(u, v))$ **fi**

$g''(k, u, v, s) =_{\mathrm{def}}$ **if** $k > n(u, v)$ **then** s

elsf $\mathrm{B}(k, u, v)$ **then** $g''(next(k, u, v), u, v, g'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), s))$

else $g''(next(k, u, v), u, v, s)$ **fi**

Syntactic constraints

KIND$[r] = \mathbf{m} \times \mathbf{p} \rightarrow \mathbf{n}$

KIND$[f, g] = \mathbf{m} \rightarrow \textbf{set of n}$

KIND$[g'] = (u\colon \mathbf{m} \times v\colon \mathbf{r} \times s\colon \textbf{set of n} \parallel \mathrm{P}(X, u, v)) \rightarrow \textbf{set of n}$

KIND$[g''] = (k\colon \mathbf{nat} \times ::: \mathbf{m} \times v\colon \mathbf{r} \times s\colon \textbf{set of n} \parallel \mathrm{P}(X, u, v)) \rightarrow \textbf{set of n}$

KIND$[next] = (\mathbf{nat} \times \mathbf{m} \times \mathbf{r}) \rightarrow \mathbf{nat}$

1.2 Some intuition and comments

The input scheme of the rule captures all kinds of functions to compute a set of values specified by a set comprehension. The output scheme gives an algorithmic way to compute the required set.

Condition (0) guarantees the computability of the solution.

In the definition of f, ZF set abstraction is used as an abbreviation for

$\{q\colon \mathbf{n} \parallel \exists\, y\colon \mathbf{p} \parallel q = r(X, y) \,\Delta\, \mathrm{Q}(x, y)\}$. Q is a kind of "input-output relation". The additional operation r allows to modify "output values" y dependent on the original input value X. In most applications r will simply be the identity on **p** (see examples below).

The purpose of the additional parameters introduced in g' is as follows:

- The parameter v of type **r** basically serves to incrementally construct elements of the set aimed at with E being kind of a neutral element to start with. Type **r** is supposed to be a "supertype" of **p** – in most applications identical with **p** or a product type with a component of type **p**.
- The parameter s of type **set of n** is used as an accumulator for the set to be constructed, and, obviously, initialized with Ø.

P is an invariant (for g' and g'') to couple the additional parameter of type **r** to the original one. To this end, condition (1) asserts the validity of the invariant for the start value E, whereas condition (5) is necessary to maintain the invariant during the computation.

Q' (in the applicability conditions) is a "generalized" form of the original predicate Q that takes the incremental construction of elements of type **r** into account. In most applications, Q' will simply be of the form

$\mathrm{Q}'(u, v, y) \equiv \exists\, y' \parallel c(v, y') = y \,\Delta\, \mathrm{Q}(u, y')$ (or $\mathrm{Q}(u, y)$ if $u = X$)

with c being a "constructor" for elements of type **r**.

T is a termination condition that serves a two-fold purpose: for the very first call of g' it has to cover the "trivial case" (i.e., the case when E is already a solution for the initial input value X); for all other recursive calls of g' it has to "signal" that the incremental construction of an element of type **r** has been completed.

Condition (3) gives the information how to construct the solution upon termination using an expression H. In most applications H will be of the simple form $H(u, v) \equiv \{r(X, v)\}$, i.e. a singleton set.

Condition (4) gives the information on how to split up the computation of the desired set into subcomputations the number of which, $n(u, v)$, may (dynamically) depend on the actual parameter values. The expressions K and R characterize the modification to the parameters within subcomputations. The expression $D(u, v)$ is relevant for cases where solutions can be computed "directly" from the parameters. In most applications (cf. below), however, it simply will be instantiated to Ø.

Condition (6) postulates the existence of a well-founded ordering on **r** or a well-founded ordering on **m** which is needed for the proof of termination (cf. below). Obviously, for any application of the rule, this condition always may be replaced by a proof of termination of the (instance of the) function g – which might be simpler in all those cases where a termination ordering different to the one in the proof of the rule is more appropriate.

Condition (7) characterizes an operation *next* for the incrementation of the "control parameter" k. *next* is characterized non-deterministically on purpose in order to be able to choose a descendant that allows pruning the computation by cutting off unsuccessful branches before they are entered. Obviously, the choice $next(k, u, v) =_{def} k+1$ is a valid descendant that satisfies condition (7). Therefore, in all applications where $next(k, u, v) =_{def} k+1$ is used, the respective proof obligation is simply ignored.

Admittedly, the whole bunch of applicability conditions looks complicated at a first glance, due to the many expressions to be "invented" and the lot of resulting proof obligations. As to the first, one should keep in mind that in most applications the simple forms as commented above are sufficient - which substantially reduces the amount of work for finding suitable instantiations. As to the latter, it will be examplified with one application below, that, whenever suitable instantiations have been found, all proofs of the instantiated applicability conditions are straightforward or even trivial.

1.3 Proof of correctness

The correctness proof of the above rule is split up into several steps each of which is proved in a calculational style. The single steps of the respective calculations are labelled for later reference. For the details of the transformations used in reasoning, we refer the reader to [Partsch 90].

a) First, we prove the equivalence of $f(X)$ (in the input scheme) with $g(X)$ (from the output scheme). To this end we define

g: **m** $\rightarrow$ **set of n**
$g(x) =_{\text{def}} g'(x, \mathrm{E}, \emptyset)$

and

g': (u: **m** $\times$ v: **r** $\times$ s: **set of n** $\|$ $\mathrm{P}(X, u, v)$) $\rightarrow$ **set of n**
$g'(u, v, s) =_{\text{def}} s \cup \{r(X, y) \,\|\, \mathrm{Q}'(u, v, y)\}$.

Then we calculate as follows:

$g(X)$
$\equiv$ [(a1) unfold g; cond. (1)]
$g'(X, \mathrm{E}, \emptyset)$
$\equiv$ [(a2) unfold g' ; neutrality of $\emptyset$ w.r.t. $\cup$]
$\{r(X, y) \,\|\, \mathrm{Q}'(X, \mathrm{E}, y)\}$
$\equiv$ [(a3) cond. (2)]
$\{r(X, y) \,\|\, \mathrm{Q}(X, y)\}$
$\equiv$ [(a4) fold f]
$f(X)$.

Since f has a defined value due to the general convention and condition (0), so do g and g' according to the above reasoning.

b) Next, we prove the equivalence of g' (as defined in (a)) with g' (as defined in the output scheme of the rule). For g' defined as in (a) we calculate as follows:

$g'(u, v, s)$
$\equiv$ [(b1) unfold g']
$s \cup \{r(X, y) \,\|\, \mathrm{Q}'(u, v, y)\}$
$\equiv$ [(b2) case-introduction; distributivity $\cup$ over conditional]
if $\mathrm{T}(u, v)$ **then** $s \cup \{r(X, y) \,\|\, \mathrm{Q}'(u, v, y)\}$ **else** $s \cup \{r(X, y) \,\|\, \mathrm{Q}'(u, v, y)\}$ **fi**
$\equiv$ [(b3) modification using cond. (3), (4)]
if $\mathrm{T}(u, v)$
then $s \cup \mathrm{H}(u, v)$
else $s \cup \mathrm{D}(u, v) \cup \bigcup_{i = 1..n(u, v)} \{r(X, y) \,\|\, \mathrm{B}(i, u, v) \,\Delta\, \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**
$\equiv$ [(b4) simplification according to cond. (7)]
if $\mathrm{T}(u, v)$
then $s \cup \mathrm{H}(u, v)$
else $s \cup \mathrm{D}(u, v) \cup \bigcup_{i = next(0, u, v)..n(u, v)} \{r(X, y) \,\|\, \mathrm{B}(i, u, v) \,\Delta\, \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**
$\equiv$ [(b5) abstraction]

if $\mathrm{T}(u, v)$ **then** $s \cup \mathrm{H}(u, v)$ **else** $g''(next(0, u, v), u, v, s \cup \mathrm{D}(u, v))$ **fi where**

g'': (k: **nat** $\times$ u: **m** $\times$ v: **r** $\times$ s: **set of n** $\parallel$ $\mathrm{P}(X, u, v)) \rightarrow$ **set of n**

$g''(k, u, v, s) =_{\mathrm{def}} s \cup \bigcup_{i = k..n(u, v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

c) Thirdly, we have to show that g'' (as defined in (b)) is equivalent to g'' from the output scheme of the rule. For g'' as defined in (b) we reason as follows:

$g''(k, u, v, s)$

$\equiv$ [(c1) unfold g'']

$s \cup \bigcup_{i = k..n(u, v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

$\equiv$ [(c2) case-introduction; simplification of **then**-branch]

if $k > n(u, v)$

then s

else $s \cup \bigcup_{i = k..n(u, v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**.

For the **else**-branch we reason as follows:

$s \cup \bigcup_{i = k..n(u, v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

$\equiv$ [(c3) $\cup$-split]

$s \cup \{r(X, y) \parallel \mathrm{B}(k, u, v) \Delta \mathrm{Q}'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), y)\} \cup$

$\bigcup_{i = k+1..next(k,u,v)-1} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\} \cup$

$\bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

$\equiv$ [(c4) simplification: def. $next \Rightarrow \bigcup_{i = k+1..next(k,u,v)-1} \{\ldots\} \equiv \emptyset$]

$s \cup \{r(X, y) \parallel \mathrm{B}(k, u, v) \Delta \mathrm{Q}'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), y)\} \cup$

$\bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**

$\equiv$ [(c5) case introduction; simplification]

if $\mathrm{B}(k, u, v)$

then $s \cup \{r(X, y) \parallel \mathrm{Q}'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), y)\} \cup$

$\bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

else $s \cup \bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**

$\equiv$ [(c6) fold g' - cond. (5) guarantees assertion; termination below]

if $\mathrm{B}(k, u, v)$

then $g'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), s) \cup$

$\bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$

else $s \cup \bigcup_{i = next(k,u,v)..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \Delta \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$ **fi**

$\equiv$ [(c7) fold g'' - cond. (5) guarantees assertion; termination below]

if $\mathrm{B}(k, u, v)$ **then** $g''(next(k, u, v), u, v, g'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), s))$

else $g''(next(k, u, v), u, v, s)$ **fi**.

Altogether, we have:

$$
\begin{aligned}
g''(k, u, v, s) \;=_{\text{def}}\; & \textbf{if } k > n(u, v) \quad \textbf{then } s \\
& \textbf{elsf } \mathrm{B}(k, u, v) \;\; \textbf{then } g''(next(k, u, v), u, v, g'(\mathrm{K}(k, u, v), \mathrm{R}(k, u, v), s)) \\
& \qquad\qquad\qquad\quad \textbf{else } g''(next(k, u, v), u, v, s) \;\; \textbf{fi}
\end{aligned}
$$

d) It remains to prove termination of g' and g'' which is required for the correctness of the folding steps. By our general convention, all expressions are defined. In the result obtained so far, g' can be eliminated by unfolding its calls in g and g'':

$g(X)$ **where**

$g(x) =_{\text{def}}$ **if** $\mathrm{T}(x, \mathrm{E})$ **then** $\emptyset \cup \mathrm{H}(x, \mathrm{E})$ **else** $g''(next(0, x, \mathrm{E}), x, \mathrm{E}, \emptyset \cup \mathrm{D}(x, \mathrm{E}))$

$$
\begin{aligned}
& g''(k, u, v, s) =_{\text{def}} \\
& \quad \textbf{if } k > n(u, v) \textbf{ then } s \\
& \quad \textbf{elsf } \mathrm{B}(k, u, v) \\
& \qquad \textbf{then } g''(next(k, u, v), u, v, \\
& \qquad\qquad\qquad \textbf{if } \mathrm{T}(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v)) \\
& \qquad\qquad\qquad\quad \textbf{then } s \cup \mathrm{H}(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v)) \\
& \qquad\qquad\qquad\qquad \textbf{else } g''(next(0, \mathrm{K}(i, u, v), \mathrm{R}(i, u, v)), \mathrm{K}(i, u, v), \mathrm{R}(i, u, v), s) \textbf{ fi}) \\
& \qquad \textbf{else } g''(next(k, u, v), u, v, s) \;\; \textbf{fi}
\end{aligned}
$$

Simplification and distributivity of function call over conditional (in g'') then yields

$$
\begin{aligned}
& g''(k, u, v, s) =_{\text{def}} \\
& \quad \textbf{if } k > n(u, v) \textbf{ then } s \\
& \quad \textbf{elsf } \mathrm{B}(k, u, v) \\
& \qquad \textbf{then if } \mathrm{T}(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v)) \\
& \qquad\qquad\quad \textbf{then } g''(next(k, u, v), u, v, s \cup \mathrm{H}(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v))) \\
& \qquad\qquad\quad \textbf{else } g''(next(k, u, v), u, v, \\
& \qquad\qquad\qquad\qquad g''(next(0, \mathrm{K}(i, u, v), \mathrm{R}(i, u, v)), \mathrm{K}(i, u, v), \mathrm{R}(i, u, v), s) \textbf{ fi} \\
& \qquad \textbf{else } g''(next(k, u, v), u, v, s) \;\; \textbf{fi}
\end{aligned}
$$

In this last version, termination of g is obvious provided g'' terminates. To prove termination of g'' we define the following (lexicographic) termination ordering $\angle$ on **nat** $\times$ **m** $\times$ **r** $\times$ **set of n**:

$$
\begin{aligned}
(k, u, v, s) \angle (k', u', v', s') \;\Leftrightarrow\; & v < v' \;\vee\; (v = v' \wedge n(u, v)-k < n(u, v)-k') \;\vee \\
& u < u' \;\vee\; (u = u' \wedge n(u, v)-k < n(u, v)-k')
\end{aligned}
$$

Then, for the recursive calls of g'' we have

$(next(k,u,v),u,v,s \cup \mathrm{H}(\mathrm{K}(i,u,v),\mathrm{R}(i,u,v))) \angle (k,u,v,s)$ [by def. of $\angle$ and *next*]

$(next(0, \mathrm{K}(i, u, v), \mathrm{R}(i, u, v)), \mathrm{K}(i, u, v), \mathrm{R}(i, u, v), s) \angle (k, u, v, s)$ [by cond. (6)]

$(next(k, u, v), u, v, g''(next(0,\mathrm{K}(i,u,v),\mathrm{R}(i,u,v)), \mathrm{K}(i, u, v), \mathrm{R}(i, u, v), s))$
$\angle (k, u, v, s)$ [by def. of $\angle$ and *next*].

$(next(k, u, v), u, v, s) \angle (k, u, v, s)$ [by def. of $\angle$ and *next*]

Hence, g'' terminates.

2. Some applications of the rule

In this section we deal with a couple of (non-trivial) problems to illustrate the power of our rule and how to use it. The first one, the N-Queens problem, is given in all details. For the remaining ones, the necessary instantiations and proof obligations can be found in the appendix.

2.1 The N-Queens Problem

The well-known N-Queens Problem asks for finding all ways to place N mutually nonattacking queens on a $N \times N$ chessboard.

Using the abbreviation

nsequ $=_{\text{def}}$ **sequ of nat**

for sequences to represent the positions in the different columns of the chessboard, the formalization of this problem is straightforward:

queens(N) **where**

queens: **nat** $\rightarrow$ **set of nsequ**

$queens(n) =_{\text{def}} \{sp\text{: } \mathbf{nsequ} \parallel |sp| = n \wedge ispossequ(sp, n) \wedge nconf(sp)\}$

where

ispossequ: **nsequ** $\times$ **nat** $\rightarrow$ **bool**

$ispossequ(s, n) =_{\text{def}} \forall\, i\text{: } \mathbf{nat} \parallel 1 \le i \le |s| \parallel s[i] \le n$

restricts the elements of **nsequ** to those less or equal to n, and

nconf: **nsequ** $\rightarrow$ **bool**

$nconf(s) =_{\text{def}} \forall\, (i, j\text{: } \mathbf{nat} \parallel 1 \le i, j \le |s| \wedge i \ne j) \parallel s[i] \ne s[j] \wedge \mathbf{abs}(s[i] - s[j]) \ne \mathbf{abs}(i - j)$

guarantees that all positions in the different columns are "nonattacking", i.e., do not mutually occupy the same row or the same diagonal.

Matching this specification with the input scheme of our rule yields (part of) the instantiations, viz.

f	**by**	*queens*
X	**by**	N
x	**by**	n
y	**by**	sp
$r(X, y)$	**by**	sp
$Q(x, y)$	**by**	$\|sp\| = n \wedge isposseq u(sp, n) \wedge nconf(sp)$.

If we further instantiate

g	**by**	qu
g'	**by**	qu'
g''	**by**	qu''
u	**by**	n
v	**by**	p
$P(X, u, v)$	**by**	$\|p\| \leq n \wedge isposseq u(p, n) \wedge nconf(p)$
E	**by**	[] (* the empty sequence of columns to start with *)
$Q'(u, v, y)$	**by**	$\exists\, s'$: **nsequ** $\|\| sp = p{+}{+}s' \wedge \|sp\| = n \wedge isposseq u(sp, n) \wedge nconf(sp)$ (* the existence of a possible extension of p to obtain a complete sequence of positions with the requested properties *)
$T(u, v)$	**by**	$\|p\| = n$
$H(u, v)$	**by**	$\{p\}$
$D(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	n
$B(i, u, v)$	**by**	$nc(p, k)$ **where** nc: **nsequ** $\times$ **nat** $\to$ **bool** $nc(p, k) =_{\text{def}} \forall\, (i$: **nat** $\|\| 1 \leq i \leq \|p\|)$: $p[i] \neq k \wedge \mathbf{abs}(p[i] - k) \neq \mathbf{abs}(i - (\|p\|+1))$ (* check whether k does not "attack" any of the elements of p *)
$K(i, u, v)$	**by**	n
$R(i, u, v)$	**by**	$p{+}{+}k$
$next(k, u, v)$	**by**	$next$: **nat** $\times$ **nat** $\times$ **nsequ** $\to$ **nat** $next(k, n, p) =_{\text{def}}$ **min** $\{i$: **nat** $\|\| k+1 \leq i \leq n + 1\ \Delta$ $\forall\, i'$: **nat** $\|\| k < i' < i \Rightarrow i' \in p\}$
WF-ORD(**r**, <)	**by**	(**nsequ**, $a < b \Leftrightarrow n - \|a\| < n - \|b\|$)

we obtain as result of the application of the rule

$qu(N)$ **where**

qu: **nat** $\to$ **set of nsequ**

$qu(n) =_{\text{def}} q'(n, [], \emptyset)$

q': **nat** $\times$ **nsequ** $\times$ **set of nsequ** $\rightarrow$ **set of nsequ**

$q'(n, p, s) =_{\text{def}}$ **if** $|p| = n$ **then** $s \cup \{p\}$ **else** $q''(next(0, n, p), n, p, s)$ **fi**

q'': **nat** $\times$ **nat** $\times$ **nsequ** $\times$ **set of nsequ** $\rightarrow$ **set of nsequ**

$q''(k, n, p, s) =_{\text{def}}$ **if** $k > n$ **then** s
elsf $nc(p, k)$ **then** $q''(next(k, n, p), n, p, q'(n, p{+}{+}k, s))$
else $q''(next(k, n, p), n, p, s)$ **fi**.

It remains to prove the instantiated applicability conditions (which is straightforward or even trivial):

(0) $(|\{sp\text{: }\mathbf{nsequ} \parallel |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\}| < \infty) \equiv \mathbf{true}$

Trivial, as already **nsequ** restricted by *isposseq u* has only finitely many elements (for any n).

(1) $(|[]| \leq n \wedge isposseq u([], n) \wedge nconf([])) \equiv \mathbf{true}$

Obviously, $|[]| = 0 \leq n$, $isposseq u([], n)$, and $nconf([])$ are all true.

(2) $|sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp) \equiv$
$\exists\, s'\text{: }\mathbf{nsequ} \parallel sp = []{+}{+}s' \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)$

Trivial, as for $s' \equiv sp$ the right-hand side expression simplifies to the one on the left-hand side.

(3) $(|p| \leq n \wedge isposseq u(p, n) \wedge nconf(p))\ \Delta\ |p| = n \equiv \mathbf{true} \vdash$
$\{p\} \equiv \{sp\text{: }\mathbf{nsequ} \parallel \exists\, s'\text{: }\mathbf{nsequ} \parallel sp = p{+}{+}s' \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\}$

From $|p| = n$, $|sp| = n$, and $sp = p{+}{+}s'$, it follows that $|s'| = 0$, and, thus, $s' = []$. Hence, the set on the right-hand side of the consequence simplifies to

$\{sp\text{: }\mathbf{nsequ} \parallel sp = p \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\}$.

The equality of both sets then follows from the premise on p.

(4) $(|p| \leq n \wedge isposseq u(p, n) \wedge nconf(p))\ \Delta\ |p| < n \equiv \mathbf{true} \vdash$
$\{sp\text{: }\mathbf{nsequ} \parallel \exists\, s'\text{: }\mathbf{nsequ} \parallel sp = p{+}{+}s' \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\} \equiv$
$\bigcup_{i = 1..n} \{sp\text{: }\mathbf{nsequ} \parallel nc(p, i)\ \Delta$
$\exists\, s'\text{: }\mathbf{nsequ} \parallel sp = (p{+}{+}i){+}{+}s' \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\}$

Due to the premise $|p| < n$ and associativity of ++, the left-hand side of the consequence can be rewritten into

$$\{sp\text{: }\mathbf{nsequ} \parallel \exists\, s'\text{: }\mathbf{nsequ},\ i\text{: }\mathbf{pos}(n) \parallel sp = (p{+}{+}i){+}{+}s' \wedge |sp| = n \wedge isposseq u(sp, n) \wedge nconf(sp)\}$$

where $\mathbf{pos}(n)$ abbreviates $(i\colon \mathbf{nat} \parallel 1 \le i \le n)$.

Since $\neg nc(p, i)$ implies $\neg nconf((p{+}{+}i){+}{+}s')$, the existential quantification on i can be restricted to those i that satisfy $nc(p, i)$. The requested equality then follows from the general rule

$$\{x \parallel \exists\, y, z \parallel \mathrm{P}(z) \wedge \mathrm{Q}(x, y, z)\} \equiv \bigcup_{z \parallel \mathrm{P}(z)} \{x \parallel \exists\, y \parallel \mathrm{Q}(x, y, z)\}.$$

(5) $1 \le i \le n \,\Delta\, (|p| \le n \wedge isposseq u(p, n) \wedge nconf(p)) \,\Delta\, |p| < n \,\Delta\, nc(p, i) \equiv \mathbf{true} \vdash$
$|p{+}{+}i| \le n \wedge isposseq u(p{+}{+}i, n) \wedge nconf(p{+}{+}i) \equiv \mathbf{true}$

$|p| < n$ implies $|p{+}{+}i| \le n$, $isposseq u(p{+}{+}i, n)$ follows immediately from the premise, and $nconf(p{+}{+}i)$ follows from $nconf(p)$ and $nc(p, i)$.

(6) $1 \le i \le n \,\Delta\, nc(p, i) \equiv \mathbf{true} \vdash n - |p{+}{+}i| < n - |p| \equiv \mathbf{true}$

The consequence is even true for arbitrary n, p, and i.

(7) $\mathbf{min}\ \{i\colon \mathbf{nat} \parallel k{+}1 \le i \le n + 1 \,\Delta\, \forall\, i'\colon \mathbf{nat} \parallel k < i' < i \Rightarrow i' \in p\} \subseteq$
$\mathbf{some}\ k'\colon \mathbf{nat} \parallel k' \in \{i\colon \mathbf{nat} \parallel k{+}1 \le i \le n{+}1 \,\Delta\, \forall\, i'\colon \mathbf{nat} \parallel k < i' < i \Rightarrow \neg nc(p, i')\}$

Obviously, $i' \in p$ implies $\neg nc(p, i')$. $\mathbf{min}\ m$ is always a descendant of $\mathbf{some}\ x \parallel x \in m$.

Remarks

- The choice of *next* as defined above (instead of the generally possible choice $next(k) =_{\mathrm{def}} k{+}1$) is a simple optimization (according to the remark on *next* in section 1).

- As a further improvement, we could have added another parameter to keep all possible candidate positions to be added to the current sequence. This would require modified instantiations, viz.

$\mathrm{P}(X, u, v)$	**by**	$\lvert p\rvert \le n \wedge isposseq u(p, n) \wedge nconf(p) \wedge \forall\, (i\colon \mathbf{pos}(n) \parallel i \in c) \parallel nc(p, i)$
E	**by**	$([], \{i\colon \mathbf{pos}(n)\})$
$\mathrm{B}(i, u, v)$	**by**	$c \ne \varnothing$
$\mathrm{R}(i, u, v)$	**by**	$(p{+}{+}i, \{k\colon \mathbf{pos}(n) \parallel nc(p{+}{+}i, k)\})$.

Together with

$next(k, c) =_{\mathrm{def}}$ **if** $c \ne \varnothing$ **then min** c **else** $n{+}1$ **fi**

these modified instantiations then would have resulted in

$qu(N)$ **where**

$qu(n) =_{\mathrm{def}} q'(n, [], \{i\colon \mathbf{pos}(n)\}, \varnothing)$

$q'(n, p, c, s) =_{\mathrm{def}}$ **if** $|p| = n$ **then** $s \cup \{p\}$ **else** $q''(next(0, c), n, p, c, s)$ **fi**

$q''(k, n, p, c, s)$ $=_{\text{def}}$
  if $k > n$ **then** s
  elsf $c \neq \emptyset$ **then** $q''(next(k, c), n, p, c, q'(n, p{+}{+}k, \{i\text{: }\mathbf{pos}(n)\} \parallel nc(p{+}{+}k, i)\}, s))$
  else $q''(next(k, c), n, p, c, s)$ **fi**.

As a further improvement – which is left to the interested reader – one could think of applying finite differencing to efficiently compute the new candidate set in the first recursive call of g''.

2.2 Permutations

The problem asks for finding all permutations of a given sequence (without duplications) of some kind of elements.

Using

ndsequ $=_{\text{def}}$ « sequences without duplicate elements »

and

same-els: **ndsequ** × **ndsequ** → **bool**
  same-els(q, sp) $=_{\text{def}}$ « q and sp have exactly the same elements »

a formalization of the problem is obvious:

perms(Q) **where**

perms: **ndsequ** → **set of ndsequ**
  perms(q) $=_{\text{def}}$ $\{sp\text{: }\mathbf{ndsequ} \parallel \textit{same-els}(q, sp)\}$.

The result of rule application is

$p(Q)$ **where**

p: **ndsequ** → **set of ndsequ**
  $p(q)$ $=_{\text{def}}$ $p'(q, [], \emptyset)$

p': **ndsequ** × **ndsequ** × **set of ndsequ** → **set of ndsequ**
  $p'(q, qp, s)$ $=_{\text{def}}$ **if** $q = []$ **then** $s \cup \{qp\}$ **else** $p''(1, q, qp, s)$ **fi**

p'': **nat** × **ndsequ** × **ndsequ** × **set of ndsequ** → **set of ndsequ**
  $p''(k, q, qp, s)$ $=_{\text{def}}$ **if** $k > |q|$ **then** s
  elsf true then $p''(k+1, q, qp, p'(q\text{-}q[k], qp{+}{+}q[k], s))$
  else $p''(k+1, q, qp, s)$ **fi**

where p'' can be further simplified to

$p''(k, q, qp, s)$ $=_{\text{def}}$ **if** $k > |q|$ **then** s **else** $p''(k+1, q, qp, p'(q\text{-}q[k], qp{+}{+}q[k], s))$ **fi**.

2.3 Segmentations (of a non-empty sequence)

The problem is as follows: Given an arbitrary non-empty sequence s (of a certain kind of elements), it is requested to compute all possible "segmentations" of s, i.e., all possible ways of cutting s into non-empty subsequences.

Using

$$\textbf{msequ} =_{\text{def}} \text{« sequences with elements of type } \textbf{m} \text{ »},$$
$$\textbf{ssequ} =_{\text{def}} (s\text{: } \textbf{sequ of msequ} \parallel \forall\ 1 \le i \le |s| \parallel s[i] \ne [\,]),$$

and

$$\mathit{flatten}\text{: } \textbf{ssequ} \to \textbf{msequ}$$
$$\mathit{flatten}(sp) =_{\text{def}} \textbf{if } sp = [\,] \textbf{ then } [\,] \textbf{ else } \textbf{hd}sp \mathbin{++} \mathit{flatten}(\textbf{tl}sp) \textbf{ fi}$$

the problem may be formalized by

$$\mathit{all\text{-}segs}(Q) \textbf{ where}$$

$$\mathit{all\text{-}segs}\text{: } \textbf{msequ} \to \textbf{set of ssequ}$$
$$\mathit{all\text{-}segs}(q) =_{\text{def}} \{sp\text{: } \textbf{ssequ} \parallel \mathit{flatten}(sp) = q\}.$$

The "direct" result of rule application (i.e., without subsequent simplifications) is

$$\mathit{segs}(Q) \textbf{ where}$$

$$\mathit{segs}\text{: } \textbf{msequ} \to \textbf{set of ssequ}$$
$$\mathit{segs}(q) =_{\text{def}} \mathit{segs}'(q, [\,], \emptyset)$$

$$\mathit{segs}'\text{: } \textbf{msequ} \times \textbf{ssequ} \times \textbf{set of ssequ} \to \textbf{set of ssequ}$$
$$\mathit{segs}'(q, qp, s) =_{\text{def}} \textbf{if } q = [\,] \textbf{ then } s \cup \{qp\} \textbf{ else } \mathit{segs}''(1, q, qp, s) \textbf{ fi}$$

$$\mathit{segs}''\text{: } \textbf{nat} \times \textbf{msequ} \times \textbf{ssequ} \times \textbf{set of ssequ} \to \textbf{set of ssequ}$$
$$\begin{aligned}&\mathit{segs}''(k, q, qp, s) =_{\text{def}}\\ &\quad \textbf{if } k > 2 \textbf{ then } s\\ &\quad \textbf{elsf true then } \mathit{segs}''(k+1, q, qp, \mathit{segs}'(\textbf{tl}\ q, \textbf{if } k{=}1 \textbf{ then } qp\mathbin{++_{\text{lst}}}[\textbf{hd}q] \textbf{ else } qp\mathbin{++}[[\textbf{hd}q]] \textbf{ fi}, s))\\ &\quad\qquad\qquad \textbf{else } \mathit{segs}''(k+1, q, qp, s) \textbf{ fi}\end{aligned}$$

where

$$\mathbin{++_{\text{lst}}}\text{: } \textbf{ssequ} \times \textbf{msequ} \to \textbf{ssequ}$$
$$qp\mathbin{++_{\text{lst}}}x =_{\text{def}} \textbf{if } qp = [\,] \textbf{ then } [x] \textbf{ else } \textbf{fst}qp\mathbin{++}(\textbf{lst}qp\mathbin{++}x) \textbf{ fi}.$$

By simplification of the outer conditional and distributivity of function call over conditional, the definition of segs'' can be further transformed into

$segs''(k, q, qp, s) =_{\text{def}}$

$\quad$ **if** $k > 2$ **then** s

$\quad$ **elsf** $k = 1$ **then** $segs''(k+1, q, qp, segs'(\mathbf{tl}\ q, qp{+}{+}_{\text{lst}}[\mathbf{hd}q], s))$

$\quad\quad\quad$ **else** $segs''(k+1, q, qp, segs'(\mathbf{tl}\ q, qp{+}{+}[[\mathbf{hd}q]], s))$ **fi**.

By successive unfoldings and simplifications, $segs''(1, q, qp, s)$ can be further transformed into

$segs'(\mathbf{tl}\ q, qp{+}{+}[[\mathbf{hd}q]], segs'(\mathbf{tl}\ q, qp{+}{+}_{\text{lst}}[\mathbf{hd}q], s))$.

Hence, $segs''$ can be eliminated and the final version of $segs'$ reads:

$segs'(q, qp, s) =_{\text{def}}$

$\quad$ **if** $q = []$ **then** $s \cup \{qp\}$ **else** $segs'(\mathbf{tl}\ q, qp{+}{+}[[\mathbf{hd}q]], segs'(\mathbf{tl}\ q, qp{+}{+}_{\text{lst}}[\mathbf{hd}q], s))$ **fi**

2.4 Partitions of a natural number (> 0)

This problem (which is closely related to the previous one) asks for finding all possible ways of representing a positive natural number as a sum.

Again, a formalization is straightforward:

$all\text{-}parts(Q)$ **where**

$all\text{-}parts$: **pnat** $\rightarrow$ **set of psequ**

$\quad all\text{-}parts(q) =_{\text{def}} \{sp\text{: } \mathbf{psequ} \parallel sum(sp) = q\}$

where

pnat $=_{\text{def}}$ (n: **nat** $\parallel n > 0$)

psequ $=_{\text{def}}$ **sequ of pnat**

and

sum: **psequ** $\rightarrow$ **nat**

$\quad sum(sp) =_{\text{def}}$ **if** $sp = []$ **then** 0 **else** $\mathbf{hd}sp + sum(\mathbf{tl}sp)$ **fi**.

The result of rule application is

$parts(Q)$ **where**

$parts$: **pnat** $\rightarrow$ **set of psequ**

$\quad parts(q) =_{\text{def}} parts'(q, [], \varnothing)$

$parts'$: **pnat** $\times$ **psequ** $\times$ **set of psequ** $\rightarrow$ **set of psequ**

$\quad parts'(q, qp, s) =_{\text{def}}$ **if** $q = 0$ **then** $s \cup \{qp\}$ **else** $parts''(1, q, qp, s)$ **fi**

$parts''$: **nat** × **pnat** × **psequ** × **set of psequ** → **set of psequ**

$parts''(k, q, qp, s) =_{def}$

if $k > 2$ **then** s

elsf true then $parts''(k+1, q, qp, parts'(q-1,$ **if** $k=1$ **then** $qp+_{lst}1$ **else** $qp{+}{+}[1]$ **fi**, $s))$

else $parts''(k+1, q, qp, s)$ **fi**

where

$+_{lst}$: **psequ** × **nat** → **psequ**

$qp+_{lst}x =_{def}$ **if** $qp = []$ **then** $[x]$ **else** $\mathbf{fst}qp{+}{+}(\mathbf{lst}qp+x)$ **fi**.

Again, $parts''$ can be further transformed (using simplification of the outer conditional and distributivity of function call over conditional) into

$parts''(k, q, qp, s) =_{def}$

if $k > 2$ **then** s

elsf $k=1$ **then** $parts''(k+1, q, qp, parts'(q-1, qp+_{lst}1, s))$

else $parts''(k+1, q, qp, parts'(q-1, qp{+}{+}[1], s))$ **fi**.

By successive unfoldings and simplifications, $parts''(1, q, qp, s)$ can be further transformed into

$parts'(q-1, qp{+}{+}[1], parts'(q-1, qp+_{lst}1, s))$.

Hence, $parts''$ can be eliminated and the final version of $parts'$ reads:

$parts'(q, qp, s) =_{def}$

if $q = 0$ **then** $s \cup \{qp\}$ **else** $parts'(q-1, qp{+}{+}[1], parts'(q-1, qp+_{lst}1, s))$ **fi**

2.4 All subsequences

The problem is as follows: Given an arbitrary sequence s (of a certain kind of elements), it is requested to compute the set of all subsequences.

Using

msequ $=_{def}$ « sequences with elements of type **m** »,

and

$issubsequ$: **msequ** × **msequ** → **bool**

$issubsequ([], t) =_{def}$ **true**

$issubsequ(x{+}{+}s, []) =_{def}$ **false**

$issubsequ(x{+}{+}s, y{+}{+}t) =_{def} (x = y \wedge issubsequ(s, t)) \vee issubsequ(x{+}{+}s, t)$

the problem may be formalized by

$all\text{-}subs(Q)$ **where**

$all\text{-}subs$: **msequ** $\rightarrow$ **set of msequ**

$all\text{-}subs(q) \;=_{\mathrm{def}}\; \{q'\text{: }\mathbf{msequ} \parallel issubssequ(q', q)\}$.

The result of rule application is

$subs(Q)$ **where**

$subs$: **msequ** $\rightarrow$ **set of msequ**

$subs(q) \;=_{\mathrm{def}}\; subs'(q, [], \emptyset)$

$subs'$: **msequ** $\times$ **msequ** $\times$ **set of msequ** $\rightarrow$ **set of msequ**

$subs'(q, p, s) \;=_{\mathrm{def}}$ **if** $q = []$ **then** $s \cup \{p\}$ **else** $subs''(1, q, p, s)$ **fi**

$subs''$: **nat** $\times$ **msequ** $\times$ **msequ** $\times$ **set of msequ** $\rightarrow$ **set of msequ**

$subs''(k, q, p, s) \;=_{\mathrm{def}}$

if $k > 2$ **then** s

elsf true then $subs''(k{+}1, q, p, subs'(\mathbf{tl}q,$ **if** $k{=}1$ **then** p **else** $p{+}{+}[\mathbf{hd}q]$ **fi**, $s))$

else $subs''(k{+}1, q, p, s)$ **fi**.

By simplification of the outer conditional and distributivity of function call over conditional, the definition of $subs''$ can be further transformed into

$subs''(k, q, p, s) \;=_{\mathrm{def}}$

if $k > 2$ **then** s

elsf $k{=}1$ **then** $subs''(k{+}1, q, p, subs'(\mathbf{tl}\ q, p, s))$

else $subs''(k{+}1, q, p, subs'(\mathbf{tl}\ q, p{+}{+}[\mathbf{hd}q], s))$ **fi**.

By successive unfoldings and simplifications, $subs''(1, q, p, s)$ can be further transformed into

$subs'(\mathbf{tl}\ q, p{+}{+}[\mathbf{hd}q], subs'(\mathbf{tl}\ q, p, s))$.

Hence, $subs''$ can be eliminated and the final version of $subs'$ reads:

$subs'(q, p, s) \;=_{\mathrm{def}}$

if $q = []$ **then** $s \cup \{p\}$ **else** $subs'(\mathbf{tl}\ q, p{+}{+}[\mathbf{hd}q], subs'(\mathbf{tl}\ q, p, s))$ **fi**

Remarks

- Obviously, the above remains correct, if we substitute **msequ** by

  ndsequ $=_{\mathrm{def}}$ « sequences without duplicate elements ».

  In this way, the above treatment also covers the problem of all subsets of a set.

– Also, the problem of computing all subsequences less than (or greater than) a given size n is captured, if we instantiate

$$\mathrm{H}(u, v) \quad \textbf{by} \quad \textbf{if } |p| < n \textbf{ then } \{p\} \textbf{ else } \emptyset \textbf{ fi}$$

and simplify afterwards.

2.6 The Pack-Problem

This problem – which is related to the one dealt with in [Partsch 90], on p. 138 and p. 256 – is a typical representative of a Knapsack problem. Given a collection of blocks (of known size) and collection of boxes (of known size), it is asked for finding all ways of associating blocks to boxes such that all blocks fit into their associated box.

Using

blocks $=_{\text{def}}$ **sequ of block**
boxes $=_{\text{def}}$ **sequ of box**
corr $=_{\text{def}}$ EMAP(**block**, **box**, =)

the problem can be formally specified by

$all\text{-}packs(K, B)$ **where**

$all\text{-}packs$: **blocks** $\times$ **boxes** $\to$ **set of corr**
$all\text{-}packs(k, b) =_{\text{def}} \{A\text{: } \mathbf{corr} \parallel legal(A, k, b)\}$

where

$legal$: **corr** $\times$ **blocks** $\times$ **boxes** $\to$ **bool**
$legal(a, k, b) =_{\text{def}} dom(a) = k \wedge b \supseteq ran(a) \wedge stowable(b, a)$
$stowable$: **boxes** $\times$ **corr** $\to$ **bool**
$stowable(b, a) =_{\text{def}} \forall (i\text{: } \mathbf{nat} \parallel 1 \le i \le |b|) \parallel (\Sigma_{k \in \{x \in dom(a) \parallel a[x] = b[i]\}} size(k)) \le size(b[i])$

guarantees that (a) all blocks are associated to a box, (b) all associated boxes are available, and (c) all blocks fit into their associated box.

The result of rule application is

$ap(K, B)$ **where**

ap: **blocks** $\times$ **boxes** $\to$ **set of corr**
$ap(k, b) =_{\text{def}} ap'(k, b, \emptyset, \emptyset)$

ap': **blocks** $\times$ **boxes** $\times$ **corr** $\times$ **set of corr** $\to$ **set of corr**
$ap'(k, b, a, s) =_{\text{def}}$ **if** $k = []$ **then** $s \cup \{a\}$ **else** $ap''(1, k, b, a, s)$ **fi**

ap'': **nat** $\times$ **blocks** $\times$ **boxes** $\times$ **corr** $\times$ **set of corr** $\rightarrow$ **set of corr**

$ap''(i, k, b, a, s) =_{\text{def}}$

if $i > |b|$ **then** s

elsf $fits(\mathbf{hd}k, b[i])$ **then** $ap''(i+1, k, b, a, ap'(\mathbf{tl}k, b[i \ominus |\mathbf{hd}k|], a[\mathbf{hd}k] \leftarrow b[i], s))$

else $ap''(i+1, k, b, a, s)$ **fi**

where

$fits$: **block** $\times$ **box** $\rightarrow$ **bool**

$fits(x, y) =_{\text{def}} size(x) \leq size(y)$

and

$\ominus$: **boxes** $\times$ **nat** $\times$ **block** $\rightarrow$ **boxes**

$b[i \ominus k] =_{\text{def}}$ « b updated such that the size of its i-th component is decreased by k ».

2.7 Topological Sortings

Given a partial ordering (by a set of pairs of elements), this problem asks for finding all total orderings of the elements (involved in the partial ordering) that "preserve" the partial ordering, i.e., whenever $a < b$ due to the partial ordering, then also $a < b$ in the total ordering.

In order to represent partial orderings, we use

pord $=_{\text{def}}$ (po: **set of** (**m**, **m**) || $\forall$ (a, b): (**m**, **m**) || $(a, b) \in po \Rightarrow (b, a) \notin po$).

The set of elements (involved in the partial ordering) is represented by

ndsequ $=_{\text{def}}$ « sequences without duplicate elements »

and obtained from the originally given partial ordering by

els: **pord** $\rightarrow$ **ndsequ**

$els(po) =_{\text{def}} set\text{-}to\text{-}sequ(\{a$: **m** || $\exists$ b: **m** || $(a, b) \in po \vee (b, a) \in po\})$

(where $set\text{-}to\text{-}sequ$ converts a set into a sequence).

With these prerequisites our problem may be formalized as follows:

$top\text{-}sorts(els(PO), PO)$ **where**

$top\text{-}sorts$: **ndsequ** $\times$ **pord** $\rightarrow$ **set of ndsequ**

$top\text{-}sorts(m, po) =_{\text{def}} \{t$: **ndsequ** || $same\text{-}els(t, m) \wedge respects(t, po)\}$

where

$same\text{-}els$: **ndsequ** $\times$ **ndsequ** $\rightarrow$ **bool**

$same\text{-}els(q, sp) =_{\text{def}}$ « q and sp have exactly the same elements » and

respects: **ndsequ** $\times$ **ndsequ** $\to$ **bool**

$respects(t, po) =_{\text{def}} \forall\ 1 \le i, j \le |t| \parallel i < j \vee (t[i], t[j]) \notin po.$

The result of rule application is

$ts(els(PO), PO)$ **where**

ts: **ndsequ** $\times$ **pord** $\to$ **set of ndsequ**

$ts(m, po) =_{\text{def}} ts'(m, po, [], \emptyset)$

ts': **ndsequ** $\times$ **pord** $\times$ **ndsequ** $\times$ **set of ndsequ** $\to$ **set of ndsequ**

$ts'(m, po, to, s) =_{\text{def}}$ **if** $m = []$ **then** $s \cup \{to\}$ **else** $ts''(1, m, po, to, s)$ **fi**

ts'': **nat** $\times$ **ndsequ** $\times$ **pord** $\times$ **ndsequ** $\times$ **set of ndsequ** $\to$ **set of ndsequ**

$s''(k, m, po, to, s) =_{\text{def}}$

if $k > |m|$ **then** s

elsf $ismin(m[k], po)$ **then** $ts''(k+1, m, po, to, ts'(m-m[k], rem(m[k], po), to{+}{+}m[k], s))$

else $ts''(k+1, m, po, to, s)$ **fi**

where

ismin: **m** $\times$ **pord** $\to$ **bool**

$ismin(x, q) =_{\text{def}} \neg\exists\ y \parallel (y, x) \in q$

and

rem: **m** $\times$ **pord** $\to$ **pord**

$rem(x, q) =_{\text{def}} q - \{(x, y) \parallel (x, y) \in q\}.$

2.8 Maximal paths in a graph

Given a finite, directed graph (by a set of nodes and – for each node – a sequence of its successors), this problem asks for finding all paths (without cycles) of maximal length starting from a given node.

In order to represent finite directed graphs, we use

graph $=_{\text{def}}$ « finite, directed graphs as defined above »,

node $=_{\text{def}}$ « nodes of a graph »,

and

nsequ $=_{\text{def}}$ (sn: **sequ of node** $\parallel sn \neq \emptyset$).

The successors of a node w.r.t. a given graph can be obtained by

$succs$: **node** $\times$ **graph** $\rightarrow$ **sequ of node**

$succs(n, g) =_{\text{def}}$ « sequence of successors of n in g ».

With these prerequisites our problem may be formalized as follows:

$all\text{-}paths(N, G)$ **where**

$all\text{-}paths$: **node** $\times$ **graph** $\rightarrow$ **set of nsequ**

$all\text{-}paths(n, g) =_{\text{def}} \{t\text{: } \mathbf{nsequ} \parallel ismaximal(t, n, g)\}$

where

$ismaximal$: **nsequ** $\times$ **node** $\times$ **graph** $\rightarrow$ **bool**

$ismaximal(t, n, g) =_{\text{def}} \mathbf{hd}t = n \wedge isend(t, g) \wedge connected(t, g) \wedge acyclic(t)$

$isend$: **nsequ** $\times$ **graph** $\rightarrow$ **bool**

$isend(t, g) =_{\text{def}} succs(\mathbf{lst}t, g) = [\,] \vee \forall\ (i\text{: } \mathbf{nat} \parallel 1 \leq i < |succs(\mathbf{lst}t, g)|) \parallel succs(\mathbf{lst}t, g)[i] \in t$

$connected$: **nsequ** $\times$ **graph** $\rightarrow$ **bool**

$connected(t, g) =_{\text{def}} \forall\ (i\text{: } \mathbf{nat} \parallel 1 \leq i < |t|) \parallel t[i+1] \in succs(t[i], g)$

and

$acyclic$: **nsequ** $\rightarrow$ **bool**

$acyclic(t) =_{\text{def}} \forall\ (i, j\text{: } \mathbf{nat} \parallel 1 \leq i, j \leq |t|) \parallel i \neq j \Rightarrow t[i] \neq t[j]$.

The result of rule application is

$paths(N, G)$ **where**

$paths$: **node** $\times$ **graph** $\rightarrow$ **set of nsequ**

$paths(n, g) =_{\text{def}} paths'(n, g, [n], \emptyset)$

$paths'$: **node** $\times$ **graph** $\times$ **nsequ** $\times$ **set of nsequ** $\rightarrow$ **set of nsequ**

$paths'(n, g, ns, s) =_{\text{def}}$ **if** $isend(ns, g)$ **then** $s \cup \{ns\}$ **else** $paths''(1, n, g, ns, s)$ **fi**

$paths''$: **nat** $\times$ **node** $\times$ **graph** $\times$ **nsequ** $\times$ **set of nsequ** $\rightarrow$ **set of nsequ**

$paths''(k, n, g, ns, s) =_{\text{def}}$

if $k > |succs(\mathbf{lst}ns, g)|$ **then** s

elsf $succs(\mathbf{lst}ns, g)[k] \notin ns$ **then** $paths''(k+1, n, g, ns, paths'(n, g, ns{+}{+}succs(\mathbf{lst}ns, g)[k], s))$

else $paths''(k+1, n, g, ns, s)$ **fi**.

2.9 A Prolog Interpreter

A Prolog interpreter takes a Prolog program and a query and yields as a result either the set of all substitutions (for the variables in the query) such that the instantiated query is a logical consequence from the program, or it does not terminate. If there is no substitution (such that the instantiated query follows from the program) the result is the empty set.

In order to formally specify the problem, we first have to define the notions of program, query, and substitution. A program is a sequence of clauses (or rules) each of which consists of a literal as its left-hand side and a (possibly empty) sequence of literals as its right-hand side. A literal in turn consists of a predicate symbol and a (possibly empty) sequence of terms (as arguments). And a term is either a constant symbol, a variable symbol, or a function symbol together with a sequence of terms as arguments. All this can straightforwardly be formalized as follows:

program $=_{\text{def}}$ **sequ of clause**
clause $=_{\text{def}}$ (*lhs*: **literal**, *rhs*: **sequ of literal**)
literal $=_{\text{def}}$ (*ps*: **predsymb**, *st*: **sequ of term**)
term $=_{\text{def}}$ **const | var | (functsymb, sequ of term)**.

A query is a sequence of literals, and a substitution is an association of variable symbols with terms. Again, formalization is straightforward:

query $=_{\text{def}}$ **sequ of literal**
subst $=_{\text{def}}$ EMAP(**var**, **term**, =).

In order to construct a substitution and simultaneously to prove that the instantiated query gl follows from the program p, a Prolog interpreter uses sld-resolution which is captured by the following derivability relation:

$\underset{p}{\rightarrow}$: **query** $\times$ **subst** $\times$ **query** $\times$ **subst** $\rightarrow$ **bool**
$(gl, \Theta) \underset{p}{\rightarrow} (gl', \Theta \circ \Theta') \quad =_{\text{def}}$
$\exists\, c \in p \parallel gl' = (rhs(c') \mathbin{++} \mathbf{tl}gl)\Theta' \wedge c' = rn(c, gl) \wedge unify(\mathbf{hd}gl, lhs(c')) \neq \mathbf{fail}\ \Delta$
$\Theta' = unify(\mathbf{hd}gl, lhs(c'))$

where

$\Theta \circ \Theta'$ denotes the composition of the substitutions Θ and Θ',
$a\Theta$ denotes the application of the substitution Θ to a,
rn: **clause** $\times$ **query** $\rightarrow$ **clause**
$rn(c, gl) =_{\text{def}}$ « clause c with all variables renamed such that they do not occur in gl »

and (with c_i and v_i denoting constant and variable symbols, receptively, f, g denoting function symbols, and r_i, s_i denoting terms)

struct $=_{\text{def}}$ **term | literal**

unify: **struct** $\times$ **struct** $\rightarrow$ **subst | fail**
$unify(c_1, c_2) =_{\text{def}}$ **if** $c_1 = c_2$ **then** $\emptyset$ **else fail fi**
$unify(c_1, v_2) =_{\text{def}}$ (c_1 **for** v_2)

$unify(c_1, g(r_1,..., r_k)) =_{\text{def}}$ **fail**

$unify(v_1, c_2) =_{\text{def}} (c_2$ **for** $v_1)$

$unify(v_1, v_2) =_{\text{def}}$ **if** $v_1 \neq v_2$ **then** $(v_2$ **for** $v_1)$ **else** $\emptyset$ **fi**

$unify(v_1, g(r_1,..., r_k)) =_{\text{def}}$

if $notoccurs(v_1, g(r_1,..., r_k))$ **then** $(g(r_1,..., r_k)$ **for** $v_1)$ **else fail fi**

$unify(f(s_1,..., s_n), c_2) =_{\text{def}}$ **fail**

$unify(f(s_1,..., s_n), v_2) =_{\text{def}}$

if $notoccurs(v_2, f(s_1,..., s_n))$ **then** $(f(s_1,..., s_n)$ **for** $v_2)$ **else fail fi**

$unify(f(s_1,..., s_n), g(r_1,..., r_k)) =_{\text{def}}$

if $f = g \wedge n = k$ **then** $unifylist([s_1,..., s_n], [r_1,..., r_k], \emptyset)$ **else fail fi**

notoccurs: **var** $\times$ **struct** $\rightarrow$ **bool**

$notoccurs(v, s) =_{\text{def}}$ « v does not occur in s »

unifylist: **sequ of struct** $\times$ **sequ of struct** $\times$ **subst** $\rightarrow$ **subst | fail**

$unifylist([], [], \Theta) =_{\text{def}} \Theta$

$unifylist(g{+}{+}gl, h{+}{+}hl, \Theta) =_{\text{def}}$

if $\Theta' \neq$ **fail then** $unifylist(gl\Theta', hl\Theta', \Theta \circ \Theta')$ **else fail fi**

where $\Theta' = unify(g, h)$

Equipped with this environment, a formal specification of a Prolog interpreter is as follows:

$interprete(P, A)$ **where**

interprete: **program** $\times$ **query** $\rightarrow$ **set of subst**

$interprete(p, gl) =_{\text{def}} \{\Theta |_{vars(A)} \parallel (gl, \emptyset) \underset{p}{\rightarrow}^{*} ([], \Theta)\}$

where

$\Theta |_{vars(A)}$ denotes Θ restricted to those variable symbols occurring in A

and

$\underset{p}{\rightarrow}^{*}$ denotes the reflexive transitive closure of $\underset{p}{\rightarrow}$.

Correct application of our transformation rule requires to satisfy the applicability condition

$(|\{\Theta |_{vars(A)} \parallel (gl, \emptyset) \underset{p}{\rightarrow}^{*} ([], \Theta)\}| < \infty) \equiv$ **true**.

Whereas in the previous examples, the corresponding proof obligation was always straightforward to be shown, it is less trivial here. Indeed, it only can be proved if both the program and the query satisfy additional constraints the detailed treatment of which is beyond the scope of this paper. A similar remark holds for condition (6) about the existence of a suitable well-founded ordering. Therefore we simply take these applicability conditions for granted and yield as result of rule application:

$int(P, A)$ **where**

int: **program** $\times$ **query** $\rightarrow$ **set of subst**

$int(p, gl) =_{\text{def}} int'(p, gl, \emptyset, \emptyset)$

int': **program** $\times$ **query** $\times$ **subst** $\times$ **set of subst** $\rightarrow$ **set of subst**

$int'(p, gl, \Phi, s) =_{\text{def}}$ **if** $gl = []$ **then** $s \cup \{\Phi\, |_{vars(A)}\}$ **else** $int''(next(0, p, \mathbf{hd}gl), p, gl, \Phi, s)$ **fi**

int'': **nat** $\times$ **program** $\times$ **query** $\times$ **subst** $\times$ **set of subst** $\rightarrow$ **set of subst**

$int''(k, p, gl, \Phi, s) =_{\text{def}}$

if $k > |p|$ **then** s

elsf $\Phi' \neq$ **fail**

then $int''(next(k, p, \mathbf{hd}gl), p, gl, int'(p, (rhs(rn(p[k], gl)) ++ \mathbf{tl}gl)\Phi', \Phi \circ \Phi', s))$

else $int''(next(k, p, \mathbf{hd}gl), p, gl, \Phi, s)$ **fi where** $\Phi' = unify(\mathbf{hd}gl, lhs(rn(p[k], gl)))$

$next$: **nat** $\times$ **literal** $\rightarrow$ **nat**

$next(k, p, l) =_{\text{def}}$ **min**$\{i$: **nat** $\|$ $(k+1 \leq i \leq |p| \,\Delta\, ps(lhs(p[i])) = ps(l)) \,\nabla\, i = |p|+1\}$

2.10 Further examples

Of course, there are many more problems our rule might be applied to. These range e.g. from mathematical problems (e.g., all subsets of a set, all partitions of a set, all subsets of a set up to a fixed size, cf. remark at the end of section 2.5), combinatorial problems (e.g., space-filling puzzles, other known problems on the chessboard), or other graph problems.

3. Variants of the general rule and their applications

In the following we discuss a couple of "variants" of our general rule and illustrate them by respective examples. These "variants" look very similar to our general rule, although they deal with different kinds of specifications and require (slightly) different applicability conditions. Using the word "variant" is simply motivated by the fact that the proofs of these rules only marginally deviate from the proof of our general rule.

3.1 Variant 1: set defined by a union

3.1.1 The rule

$f(X)$ **where**

$f(x) =_{\text{def}} \bigcup_{z \in \{y \,\|\, Q(x, y)\}} r(X, z)$

$$(0)\quad (|\bigcup_{z \in \{y \,\|\, Q(x,y)\}} r(X, z)| < \infty) \equiv \mathbf{true}$$

(1), (2), (5), (6), (7) as in general rule

$$(3)\quad P(X, u, v) \,\Delta\, T(u, v) \equiv \mathbf{true} \vdash H(u, v) \equiv \bigcup_{z \in \{y \,\|\, Q'(u, v, y)\}} r(X, z)$$

$$(4)\quad P(X, u, v) \,\Delta\, \neg T(u, v) \equiv \mathbf{true} \vdash$$
$$\bigcup_{z \in \{y \,\|\, Q'(u, v, y)\}} r(X, z) \equiv$$
$$\bigcup_{i = 1..n(u, v)} \bigcup_{z \in \{y \,\|\, B(i, u, v) \,\Delta\, Q'(K(i, u, v), R(i, u, v), y)\}} r(X, z)$$

$g(X)$ **where**

$g(x) =_{\text{def}} g'(x, E, \emptyset)$

$g'(u, v, s) =_{\text{def}}$ **if** $T(u, v)$ **then** $s \cup H(u, v)$ **else** $g''(next(0, u, v), u, v, s)$ **fi**

$g''(k, u, v, s) =_{\text{def}}$ **if** $k > n(u, v)$ **then** s
elsf $B(k, u, v)$ **then** $g''(next(k, u, v), u, v, g'(K(k, u, v), R(k, u, v), s))$
else $g''(next(k, u, v), u, v, s)$ **fi**

Syntactic constraints

KIND[r] = $\mathbf{m} \times \mathbf{p} \rightarrow$ **set of n**
KIND[f, g] = $\mathbf{m} \rightarrow$ **set of n**
KIND[g'] = (u: $\mathbf{m} \times v$: $\mathbf{r} \times s$: **set of n** $\|$ $P(X, u, v)$) $\rightarrow$ **set of n**
KIND[g''] = (k: **nat** $\times\, u$: $\mathbf{m} \times v$: $\mathbf{r} \times s$: **set of n** $\|$ $P(X, u, v)$) $\rightarrow$ **set of n**
KIND[$next$] = (**nat** $\times\, \mathbf{m} \times \mathbf{r}$) $\rightarrow$ **nat**

Proof

Exactly as for general rule (with $g(x) =_{\text{def}} s \cup \bigcup_{z \in \{y \,\|\, Q(x,y)\}} r(X, z)$ in step (a) and $D(u, v)$ instantiated by $\emptyset$).

.

3.1.2 An application: The Coding Problem

This problem is also treated (in a different way) in [Partsch 90] (p. 132, p. 247) and reads as follows: Given a non-empty "clear word" N (over some character set V_1) and a "coding rule" P (which is a sequence of pairs consisting of a non-empty word over V_1 and a non-empty "code word", i.e. a non-empty word over a character set V_2), it is requested to compute the set of all encodings of N with respect to P.

Using

clear $=_{\text{def}}$ « clear words »
code $=_{\text{def}}$ « code words »

$\textbf{rule} =_{\mathrm{def}} (l\text{: }\textbf{clear}, r\text{: }\textbf{code})$

a possible formalization of the problem is as follows:

$encodings(N, P)$ **where**

$encodings$: $\textbf{clear} \times \textbf{sequ of rule} \rightarrow \textbf{set of code}$

$encodings(n, p) =_{\mathrm{def}} \bigcup_{a \in deco(n, p)} encode\ (a, p)$

where

$deco$: $\textbf{clear} \rightarrow \textbf{sequ of clear}$

$deco(n, p) =_{\mathrm{def}} \{a : \textbf{sequ of clear} \parallel flatten(a) = n \wedge codable(a, p)$

$encode$: $\textbf{sequ of clear} \times \textbf{rule} \rightarrow \textbf{set of code}$

$encode(a, p) =_{\mathrm{def}}$

$\{c : \textbf{code} \parallel \exists\ b : \textbf{sequ of code} \parallel$

$flatten(b) = c \wedge |b| = |a| \wedge \forall\ (i\text{: }\textbf{nat} \parallel 1 \leq i \leq |a|) \parallel \exists\ (j\text{: }\textbf{nat} \parallel 1 \leq j \leq |p|) \parallel b\ [i] = r(p[j])$

and

$codable$: $\textbf{sequ of clear} \times \textbf{sequ of rule} \rightarrow \textbf{bool}$

$codable(a, p) =_{\mathrm{def}} \forall\ (i\text{: }\textbf{nat} \parallel 1 \leq i \leq |a|) \parallel \exists\ (j\text{: }\textbf{nat} \parallel 1 \leq j \leq |p|) \parallel a[i] = l(p[j])$

$flatten$: $\textbf{sequ of code} \rightarrow \textbf{code}$

$flatten(b) =_{\mathrm{def}}$ **if** $b = []$ **then** $[]$ **else** $\textbf{hd}b \mathbin{++} flatten(\textbf{tl}b)$ **fi**

The result of rule application is

$enc(N, P)$ **where**

enc: $(\textbf{clear} \times \textbf{sequ of rule}) \rightarrow \textbf{set of code}$

$enc(n, p) =_{\mathrm{def}} enc'(n, p, \{[]\}, \emptyset)$

enc': $(\textbf{clear} \times \textbf{rule} \times \textbf{set of code} \times \textbf{set of code}) \rightarrow \textbf{set of code}$

$enc'(n, p, e, s) =_{\mathrm{def}}$ **if** $n = []$ **then** $s \cup e$ **else** $enc''(1, n, p, e, s)$ **fi**

enc': $(\textbf{nat} \times \textbf{clear} \times \textbf{rule} \times \textbf{set of code} \times \textbf{set of code}) \rightarrow \textbf{set of code}$

$enc''(k, n, p, e, s) =_{\mathrm{def}}$

if $k > |p|$ **then** s

elsf $l(p[k]) \;\dot{}\; n$ **then** $enc''(k{+}1, n, p, e, enc'(n \lrcorner l(p[k]), p, e \oplus r(p[k]), s))$

else $enc''(k{+}1, n, p, e, s)$ **fi**

where

$\dot{}$: $(\textbf{clear} \times \textbf{clear}) \rightarrow \textbf{bool}$

$a \;\dot{}\; b =_{\mathrm{def}}$ « a is an initial segment of b »

$\lrcorner$: (**clear** $\times$ **clear**) $\rightarrow$ **clear**

$a \lrcorner b$ $=_{\text{def}}$ « b with initial segment a removed »

$\oplus$: (**set of code** $\times$ **code**) $\rightarrow$ **set of code**

$a \oplus b$ $=_{\text{def}}$ $\{a' \mathbin{++} b \parallel a' \in a\}$

3.2 Variant 2: Existential quantification

3.2.1 The rule

$f(X)$ **where**

$f(x)$ $=_{\text{def}}$ $\exists\, y$: **p** $\parallel$ Q(x, y)

(1), (2), (5), (6), (7) as in general rule

(3) P(X, u, v) Δ T$(u, v) \equiv$ **true** $\vdash$ **true** $\equiv \exists$ y: **p** $\parallel$ Q'(u, v, y)

(4) P(X, u, v) Δ $\neg$T$(u, v) \equiv$ **true** $\vdash$

$\exists$ y: **p** $\parallel$ Q'$(u, v, y) \equiv$ $\exists_{i = 1..n(u, v)}$ $\exists$ y: **p** $\parallel$ B(i, u, v) Δ Q'(K(i, u, v), R(i, u, v), y)

$g(X)$ **where**

$g(x)$ $=_{\text{def}}$ $g'(x$, E, **false**)

$g'(u, v, s)$ $=_{\text{def}}$ **if** T(u, v) **then true else** $g''(next(0, u, v), u, v, s)$ **fi**

$g''(k, u, v, s)$ $=_{\text{def}}$ **if** $s \vee k > n(u, v)$ **then** s
elsf B(k, u, v) **then** $g''(next(k, u, v), u, v, g'(K(k, u, v)$, R$(k, u, v), s))$
else $g''(next(k, u, v), u, v, s)$ **fi**

Syntactic constraints

KIND$[f, g]$ = **m** $\rightarrow$ **bool**

KIND$[g']$ = (u: **m** $\times$ v: **r** $\times$ s: **bool** $\parallel$ P(X, u, v)) $\rightarrow$ **bool**

KIND$[g'']$ = (k: **nat** $\times$ u: **m** $\times$ v: **r** $\times$ s: **bool** $\parallel$ P(X, u, v)) $\rightarrow$ **bool**

KIND$[next]$ = (**nat** $\times$ **m** $\times$ **r**) $\rightarrow$ **nat**

Proof (changes apart from instantiating r by the identity and D(u, v) by $\emptyset$)

a) $g(x)$ $=_{\text{def}}$ $s \vee \exists\, y$: **p** $\parallel$ Q(x, y)

(a2'): unfold g' ; neutrality of **false** w.r.t. $\vee$

b) (b2'): case-introduction; distributivity $\vee$ over conditional

c) (c3'): $\vee$-split in **else**-branch

3.2.2 An Application: Top-down recognition (for context-free grammars)

Given a context-free grammar $G = (N, T, Z, P)$ (where N is a non-empty set of nonterminal symbols, T a non-empty set of terminal symbols which is disjoint from N, $Z \in N$, and P is a non-empty set of context-free productions consisting of a nonterminal symbol as its left-hand side and a (possibly empty) sequence of (nonterminal or terminal) symbols as its right-hand side) and a terminal word w, it is requested to check whether w is derivable from Z using P.

Using

nont $=_{\text{def}}$ « nonterminal symbols »
term $=_{\text{def}}$ « terminal symbols »
symb $=_{\text{def}}$ **nont | term**
prod $=_{\text{def}}$ (*lhs*: **nont**, *rhs*: **sequ of symb**)

the problem can be formalized by

recognize(P, W) **where**

recognize: **sequ of prod** $\times$ **sequ of term** $\rightarrow$ **bool**
recognize(p, w) $=_{\text{def}}$ $\exists\, y$: **sequ of term** $\parallel [Z] \underset{p}{\rightarrow}{}^{*}\, y \wedge y = w$

where

$\underset{p}{\rightarrow}$: **sequ of symb** $\times$ **sequ of symb** $\rightarrow$ **bool**
$x \underset{p}{\rightarrow} y$ $=_{\text{def}}$ $\exists\, l, r$: **sequ of symb**, i: **nat** $\parallel x = l{+}{+}lhs(p[i]){+}{+}r \wedge y = l{+}{+}rhs(p[i]){+}{+}r$

and

$\underset{p}{\rightarrow}{}^{*}$ denotes the reflexive transitive closure of $\underset{p}{\rightarrow}$.

The result of rule application is

rec(P, W) **where**

rec: **sequ of prod** $\times$ **sequ of term** $\rightarrow$ **bool**
rec(p, w) $=_{\text{def}}$ *rec'*$(p, w, [Z]$, **false**)

rec': **sequ of prod** $\times$ **sequ of term** $\times$ **sequ of symb** $\times$ **bool** $\rightarrow$ **bool**
rec'(p, w, v, s) $=_{\text{def}}$ **if** $w = [\,] \wedge v = [\,]$ **then true else** *rec''*$(next(1, p, v), p, w, v, s)$ **fi**

rec'': **nat** $\times$ **sequ of prod** $\times$ **sequ of term** $\times$ **sequ of symb** $\times$ **bool** $\rightarrow$ **bool**

$rec''(k, p, w, v, s) =_{\text{def}}$

if $s \vee k > |p|+1$ **then** s

elsf $(k \leq |p| \wedge v \neq [] \,\Delta\, \mathbf{hd}v = lhs(p[k])) \vee (k = |p|+1 \wedge w \neq [] \wedge v \neq [] \,\Delta\, \mathbf{hd}w = \mathbf{hd}v)$

then $rec''(next(k, p, v), p, w, v, rec'(p,$ **if** $k \leq |p|$ **then** w **else tl**w **fi**,

if $k \leq |p|$ **then** $rhs(p[k])$++**tl**v **else tl**v **fi**, $s))$

else $rec''(next(k, p, v), p, w, v, s)$ **fi**

$next$: **nat** $\times$ **sequ of prod** $\times$ **sequ of symb** $\rightarrow$ **nat**

$next(k, p, v) =_{\text{def}}$ **min** $\{i$: **nat** $\| (k+1 \leq i \leq |p| \,\Delta\, lhs(p[i]) = \mathbf{hd}v) \,\nabla\, i = |p|+1\}$

where rec'' can be further transformed into

$rec''(k, p, w, v, s) =_{\text{def}}$

if $s \vee k > |p|+1$ **then** s

elsf $k \leq |p| \wedge v \neq [] \,\Delta\, \mathbf{hd}v = lhs(p[k])$

then $rec''(next(k, p, v), p, w, v, rec'(p, w, rhs(p[k])$++**tl**$v, s))$

elsf $k = |p|+1 \wedge w \neq [] \wedge v \neq [] \,\Delta\, \mathbf{hd}w = \mathbf{hd}v$

then $rec''(next(k, p, v), p, w, v, rec'(p, \mathbf{tl}w, \mathbf{tl}v, s))$

else $rec''(next(k, p, v), p, w, v, s)$ **fi**.

Similar to the Prolog interpreter, proving termination is slightly more difficult than in most of the other examples. In fact, as is known, absence of left-recursion in the grammar is necessary for the proof of termination.

Of course, we could have chosen an even "stronger" definition for *next*, viz.

$next(k, p, v) =_{\text{def}}$

min $\{i$: **nat** $\| (k+1 \leq i \leq |p| \,\Delta\, lhs(p[i]) = \mathbf{hd}v \,\Delta\, \exists\, z$: **symb** $\| \mathbf{hd}v \underset{p}{\Rightarrow} \mathbf{hd}w$++$z) \,\nabla\, i = |p|+1\}$

which obviously reflects the idea of LL(1) recognition.

3.3 Variant 3: Non-deterministic choice

3.3.1 The rule

$f(X)$ **where**

$f(x) =_{\text{def}}$ **if** $\{y$: **p** $\| Q(x, y)\} \neq \emptyset$ **then some** y: **p** $\| Q(x, y)$ **else dummy fi**

(1), (2), (5), (6), (7) as in general rule

(3) $\quad P(X, u, v) \,\Delta\, T(u, v) \equiv \mathbf{true} \vdash H(u, v) \in \{y \,\|\, Q'(u, v, y)\} \equiv \mathbf{true}$

(4) $\quad P(X, u, v) \,\Delta\, \neg T(u, v) \equiv \mathbf{true} \vdash$

$\quad\quad \mathbf{some}\ y\!: \mathbf{p} \,\|\, Q'(u, v, y) \equiv\ \mathbf{some}\ y\!: \mathbf{p} \,\|\, \exists_{i = 1..n(u,v)}\, (B(i, u, v) \,\Delta\, Q'(K(i, u, v), R(i, u, v), y))$

$g(X)$ **where**

$\quad g(x) \ =_{\mathrm{def}}\ g'(x, E, \mathbf{dummy})$

$\quad g'(u, v, s) \ =_{\mathrm{def}}\ \mathbf{if}\ T(u, v)\ \mathbf{then}\ H(u, v)\ \mathbf{else}\ g''(next(0, u, v), u, v, s)\ \mathbf{fi}$

$\quad g''(k, u, v, s) \ =_{\mathrm{def}}\ \mathbf{if}\ s \neq \mathbf{dummy} \vee k > n(u, v)\ \mathbf{then}\ s$

$\quad\quad\quad \mathbf{elsf}\ B(k, u, v)\ \mathbf{then}\ g''(next(k, u, v), u, v, g'(K(k, u, v), R(k, u, v), s))$

$\quad\quad\quad\quad \mathbf{else}\ g''(next(k, u, v), u, v, s)\ \ \mathbf{fi}$

Syntactic constraints

$\quad \mathsf{KIND}[f, g] = \mathbf{m} \to \mathbf{n} \mid \mathbf{dummy}$

$\quad \mathsf{KIND}[g'] = (u\!: \mathbf{m} \times v\!: \mathbf{r} \times s\!: \mathbf{n} \mid \mathbf{dummy} \,\|\, P(X, u, v)) \to \mathbf{n} \mid \mathbf{dummy}$

$\quad \mathsf{KIND}[g''] = (k\!: \mathbf{nat} \times u\!: \mathbf{m} \times v\!: \mathbf{r} \times s\!: \mathbf{n} \mid \mathbf{dummy} \,\|\, P(X, u, v)) \to \mathbf{n} \mid \mathbf{dummy}$

$\quad \mathsf{KIND}[next] = (\mathbf{nat} \times \mathbf{m} \times \mathbf{r}) \to \mathbf{nat}$

Proof (changes apart from instantiating r by the identity and $D(u, v)$ by $\emptyset$)

a) $g(x) \ =_{\mathrm{def}}\ \mathbf{dummy} \copyright\ \mathbf{some}\ y\!: \mathbf{p} \,\|\, Q(x, y)$

where

$\quad a \copyright b \ =_{\mathrm{def}}\ \mathbf{if}\ def(a)\ \mathbf{then}\ a\ []\ def(b)\ \mathbf{then}\ b\ \mathbf{else}\ \mathbf{dummy}\ \mathbf{fi}\ \mathbf{where}$

$\quad def(\mathbf{some}\ y \,\|\, Q) =_{\mathrm{def}}\ \{y \,\|\, Q\} \neq \emptyset$

$\quad def(x) =_{\mathrm{def}}\ x \neq \mathbf{dummy}.$

(a2'): unfold g'

(a3'): cond. (2); unfold ©

b) (b2'): case-introduction

(b3'): $\supseteq$ for $\equiv$

c) (c2'): $\supseteq$ for $\equiv$

(c3'): $\vee$-split in **else**-branch

(c5'): case introduction in **else**-branch;

simplification using $\mathbf{some}\ y \,\|\, B \vee C\ \equiv\ \mathbf{some}\ y \,\|\, B \copyright\ \mathbf{some}\ y \,\|\, C$

3.3.2 An Application: Shift-Reduce parsing (of context-free grammars)

Given a context-free grammar G (as defined in the previous example) and a terminal word w, it is asked for computing a sequence of shift and reduce actions the application of which allows to reduce w to the axiom Z of G provided such a sequence exists; if not, the result should be **error**.

For a formal specification of this problem we define **nont**, **term**, **symb**, and **prod** as in the previous example. The sequence of shift and reduce actions is defined by

srsequ $=_{\text{def}}$ « sequences of S (= *shift*) and R(X, α) (= *reduce with* (X, α)) ».

The application of such a sequence to a terminal word is specified by

apply: **sequ of prod** $\times$ **sequ of symb** $\times$ **sequ of term** $\rightarrow$ **sequ of symb** $\times$ **sequ of term**

$apply([], st, w) =_{\text{def}} (st, w)$

$apply(\text{S}{+}{+}srs, st, t{+}{+}w) =_{\text{def}} apply(srs, st{+}{+}t, w)$

$apply(\text{R}(\text{X}, \alpha){+}{+}srs, st{+}{+}\alpha, w) =_{\text{def}} apply(srs, st{+}{+}\text{X}, w)$.

With these prerequisites, the original problem may be specified as follows:

parse(P, [],W) **where**

parse: **sequ of prod** $\times$ **sequ of symb** $\times$ **sequ of term** $\rightarrow$ **srsequ | error**

$parse(p, st, w) =_{\text{def}}$ **if** $\{y$: **srsequ** $\Vert\ apply(y, st, w) = ([Z], [])\} \neq \emptyset$

then some y: **srsequ** $\Vert\ apply(y, st, w) = ([Z], [])$ **else error fi**.

The result of rule application is

par(P, [],W) **where**

par: **sequ of prod** $\times$ **sequ of symb** $\times$ **sequ of term** $\rightarrow$ **srsequ | error**

$par(p, st, w) =_{\text{def}} par'(p, st, w, [],$ **error**$)$

par': **sequ of prod** $\times$ **sequ of symb** $\times$ **sequ of term** $\times$ **srsequ** $\times$ **srsequ | error** $\rightarrow$
srsequ | error

$par'(p, st, w, sr, s) =_{\text{def}}$ **if** $w = [] \wedge st = [Z]$ **then** sr **else** $par''(1, p, st, w, sr, s)$ **fi**

par'': $\textbf{nat} \times \textbf{sequ of prod} \times \textbf{sequ of symb} \times \textbf{sequ of term} \times \textbf{srsequ} \times \textbf{srsequ} \mid \textbf{error} \rightarrow$
$\textbf{srsequ} \mid \textbf{error}$

$par''(k, p, st, w, sr, s) =_{\text{def}}$
if $s \neq$ **error** $\vee\ k > |p|+1$ **then** s
elsf $(k \leq |p| \wedge \exists\ s' \parallel st = s'{+}{+}rhs(p[k])) \vee (k = |p|+1 \wedge w \neq [])$
then $par''(k+1, p, st, w, sr,$
$par'(p,$ **if** $k \leq |p|$ **then** $(s'{+}{+}lhs(p[k]), w)$ **else** $(st{+}{+}\textbf{hd}w, \textbf{tl}w)$ **fi**,
if $k \leq |p|$ **then** $sr{+}{+}\mathrm{R}(p[k])$ **else** $sr{+}{+}\mathrm{S}$ **fi**, $s))$
else $par''(k+1, p, st, w, sr, s)$ **fi**

where par'' can be further transformed into

$par''(k, p, st, w, sr, s) =_{\text{def}}$
if $s \neq$ **error** $\vee\ k > |p|+1$ **then** s
elsf $k \leq |p| \wedge \exists\ s' \parallel st = s'{+}{+}rhs(p[k])$
then $par''(k+1, p, st, w, sr, par'(p, s'{+}{+}lhs(p[k]), w, sr{+}{+}\mathrm{R}(p[k]), s))$
elsf $k = |p|+1 \wedge w \neq []$
then $par''(k+1, p, st, w, sr, par'(p, st{+}{+}\textbf{hd}w, \textbf{tl}w, sr{+}{+}\mathrm{S}, s))$
else $par''(k+1, p, st, w, sr, s)$ **fi**.

Again, as in the case of top-down recognition, additional constraints on the grammar are necessary for the proof of termination.

3.4 Variant 4: without incremental construction of the elements of the solution

3.4.1 The rule

$f(X)$ **where**
$f(x) =_{\text{def}} \{r(X, y)\text{: } \mathbf{n} \parallel \mathrm{Q}(x, y)\}$

(0) as in general rule
(3) $\mathrm{T}(u) \equiv \textbf{true} \vdash \mathrm{H}(u) \equiv \{r(X, y) \parallel \mathrm{Q}(u, y)\}$
(4) $\neg\mathrm{T}(u, v) \equiv \textbf{true} \vdash$
$\{r(X, y) \parallel \mathrm{Q}(u, y)\} \equiv \mathrm{D}(u) \cup \bigcup_{i = 1..n(u)} \{r(X, y) \parallel \mathrm{B}(i, u)\ \Delta\ \mathrm{Q}(\mathrm{K}(i, u), y)\}$
(6) $1 \leq i \leq n(u)\ \Delta\ \mathrm{B}(i, u) \equiv \textbf{true} \vdash$
$(\mathrm{K}(i, u) < u) \equiv \textbf{true}$ **where** $\text{WF-ORD}(\mathbf{m}, <)$
(7) $next(k, u) \subseteq$
some k': **nat** $\parallel k' \in \{i\text{: } \textbf{nat} \parallel k+1 \leq i \leq n(u)+1\ \Delta\ \forall\ i'\text{: } \textbf{nat} \parallel k < i' < i \Rightarrow \neg\mathrm{B}(i', u)\}$

$g(X)$ **where**

$g(x) =_{\text{def}} g'(x, \emptyset)$

$g'(u, s) =_{\text{def}}$ **if** $T(u)$ **then** $s \cup H(u)$ **else** $g''(next(0, u), u, s \cup D(u))$ **fi**

$g''(k, u, s) =_{\text{def}}$ **if** $k > n(u)$ **then** $s \cup D(u)$
elsf $B(k, u)$ **then** $g''(next(k, u), u, g'(K(k, u), s))$
else $g''(next(k, u), u, s)$ **fi**

Syntactic constraints

KIND$[r]$ = **m** × **p** → **n**
KIND$[f, g]$ = **m** → **set of n**
KIND$[g']$ = (**m** × **set of n**) → **set of n**
KIND$[g'']$ = (**nat** × **m** × **set of n**) → **set of n**
KIND$[next]$ = (**nat** × **m**) → **nat**

Proof (changes)

a) $g(x) =_{\text{def}} g'(x, \emptyset)$;
$g'(u, s) =_{\text{def}} s \cup \{r(X, y)\text{: } \mathbf{n} \parallel Q(x, y)\}$;
(a1'): unfold g
(a3'): –

3.4.2 An Application: Reachable Nodes in graph

Given a finite, directed graph (by a set of nodes and – for each node – a sequence of its successors), this problem asks for finding the set of all nodes reachable from a given node.

In order to represent finite directed graphs, we use (as in section 2.8)

graph $=_{\text{def}}$ « finite, directed graphs as defined above »,
node $=_{\text{def}}$ « nodes of a graph ».

The successors of a node w.r.t. a given graph can be obtained by

succs: **node** × **graph** → **sequ of node**
$succs(n, g) =_{\text{def}}$ « sequence of successors of n in g ».

With these prerequisites our problem may be formalized as follows:

$reachables(N, G)$ **where**

reachables: **node** × **graph** → **set of node**
$reachables(n, g) =_{\text{def}} \{m\text{: } \mathbf{node} \parallel reachable(m, n, g)\}$

where

$reachable$: **node** $\times$ **node** $\times$ **graph** $\rightarrow$ **bool**

$reachable(m, n, g) =_{\text{def}}$

$m = n \ \vee \exists\ t$: **sequ of node** $\|\ t \neq [\,]\ \Delta$ (**hd**$t = n \wedge$ **lst**$t = m \wedge connected(t, g)$

and

$connected$: **nsequ** $\times$ **graph** $\rightarrow$ **bool**

$connected(t, g) =_{\text{def}} \forall\ (i$: **nat** $\|\ 1 \leq i < |t|)\ \|\ t[i+1] \in succs(t[i], g)$.

The result of rule application is

$r(N, G)$ **where**

r: **node** $\times$ **graph** $\rightarrow$ **set of node**

$r(n, g) =_{\text{def}} r'(n, g, \varnothing)$

r': **node** $\times$ **graph** $\times$ **set of node** $\rightarrow$ **set of node**

$r'(n, g, s) =_{\text{def}}$ **if** $succs(n, g) = [\,]$ **then** $s \cup \{n\}$ **else** $r''(1, n, g, s \cup \{n\})$ **fi**

r'': **nat** $\times$ **node** $\times$ **graph** $\times$ **set of node** $\rightarrow$ **set of node**

$paths''(k, n, g, s) =_{\text{def}}$

if $k > |succs(n, g)|$ **then** s

elsf $\neg visited(g, succs(n, g)[k])$ **then** $r''(k+1, n, g, r'(succs(n, g)[k], vis(n, g), s))$

else $r''(k+1, n, g, s)$ **fi**

where

$visited$: **graph** $\times$ **node** $\rightarrow$ **bool**

$visited(g, n) =_{\text{def}}$ « n marked as visited in g »

and

vis: **node** $\times$ **graph** $\rightarrow$ **graph**

$vis(n, g) =_{\text{def}}$ « g with n marked as visited ».

3.5 Variant 5: A special instance of the rule

3.5.1 The rule

Differently to the previous variants, we now consider a variant that is obtained by specializing the general rule by

- defining $n(u, v) =_{\text{def}} 1$ and $next(k, u, v) =_{\text{def}} k+1$ and instantiating these definitions
- simplifying the resulting expressions, and

- skipping the dependencies (on i) of the expressions K, R, and B.

Thus, we get the following specialization of our general rule:

$f(X)$ **where**

$f(x) =_{\text{def}} \{r(X, y)\text{: } \mathbf{n} \parallel \mathrm{Q}(x, y)\}$

(0), (1), (2), (3) as in general rule

(4) $\mathrm{P}(X, u, v) \,\Delta\, \neg\mathrm{T}(u, v) \equiv \mathbf{true} \vdash$
$\{r(X, y) \parallel \mathrm{Q}'(u, v, y)\} \equiv \{r(X, y) \parallel \mathrm{B}(u, v) \,\Delta\, \mathrm{Q}'(\mathrm{K}(u, v), \mathrm{R}(u, v), y)\}$

(5) $\mathrm{P}(X, u, v) \,\Delta\, \neg\mathrm{T}(u, v) \,\Delta\, \mathrm{B}(u, v) \equiv \mathbf{true} \vdash$
$\mathrm{P}(X, \mathrm{K}(u, v), \mathrm{R}(u, v)) \equiv \mathbf{true}$

(6) $\mathrm{B}(u, v) \equiv \mathbf{true} \vdash$ $(\mathrm{R}(u, v) < v) \equiv \mathbf{true}$ **where** WF-ORD(r, <) **or**
$(\mathrm{K}(u, v) < u) \equiv \mathbf{true}$ **where** WF-ORD(m, <)

$g(X)$ **where**

$g(x) =_{\text{def}} g'(x, \mathrm{E}, \emptyset)$

$g'(u, v, s) =_{\text{def}}$ **if** $\mathrm{T}(u, v)$ **then** $s \cup \mathrm{H}(u, v)$ **else** $g''(1, u, v, s)$ **fi**

$g''(k, u, v, s) =_{\text{def}}$ **if** $k > 1$ **then** s
elsf $\mathrm{B}(u, v)$ **then** $g''(k+1, u, v, g'(\mathrm{K}(u, v), \mathrm{R}(u, v), s))$
else $g''(k+1, u, v, s)$ **fi**

Syntactic constraints

as in general rule

A much simpler form of the output scheme may be obtained by further transformation of the output scheme:

(a) $g''(2, u, v, s)$

$\equiv$ [instantiation]

s

(b) $g''(1, u, v, s)$

$\equiv$ [unfold g'']

if $1 > 1$ **then** s
elsf $\mathrm{B}(u, v)$ **then** $g''(2, u, v, g'(\mathrm{K}(u, v), \mathrm{R}(u, v), s))$ **else** $g''(2, u, v, s)$ **fi**

$\equiv$ [simplification of conditional]

if $\mathrm{B}(u, v)$ **then** $g''(2, u, v, g'(\mathrm{K}(u, v), \mathrm{R}(u, v), s))$ **else** $g''(2, u, v, s)$ **fi**

$\equiv$ [(a)]

if $B(u, v)$ **then** $g'(K(u, v), R(u, v), s)$ **else** s **fi**

(c) $g'(u, v, s)$

$\equiv$ [unfold g']

if $T(u, v)$ **then** $s \cup H(u, v)$ **else** $g''(1, u, v, s)$ **fi**

$\equiv$ [(b)]

if $T(u, v)$ **then** $s \cup H(u, v)$

elsf $B(u, v)$ **then** $g'(K(u, v), R(u, v), s)$ **else** s **fi**

(d) Define $g'''(u, v) =_{def} g'(u, v, \emptyset)$. Then

$g'''(u, v)$

$\equiv$ [unfold g''']

$g'(u, v, \emptyset)$

$\equiv$ [unfold g' as obtained in (b)]

if $T(u, v)$ **then** $\emptyset \cup H(u, v)$

elsf $B(u, v)$ **then** $g'(K(u, v), R(u, v), \emptyset)$ **else** $\emptyset$ **fi**

$\equiv$ [simplification; fold g''']

if $T(u, v)$ **then** $H(u, v)$

elsf $B(u, v)$ **then** $g'''(K(u, v), R(u, v))$ **else** $\emptyset$ **fi**.

Altogether, we have derived as a new output scheme:

$g(X)$ **where**

g: **m** $\rightarrow$ **set of n**

$g(x) =_{def} g'''(x, E)$ **where**

g''': (u: **m** $\times$ v: **r** $\|$ $P(X, u, v)$) $\rightarrow$ **set of n**

$g'''(u, v) =_{def}$

if $T(u, v)$ **then** $H(u, v)$

elsf $B(u, v)$ **then** $g'''(K(u, v), R(u, v))$ **else** $\emptyset$ **fi**

3.5.2 An application: Prime factors of a natural number (≥ 2)

Given a natural number n (≥ 2), it is asked to compute the sequence of all prime factors of n in increasing order.

The output type may be formally specified by

psequ $=_{def}$ (s: **natsequ** $\|$ $\forall\ 1 \leq i \leq |s|$ $\|$ $prime(s[i])$)

where

prime: **nat** $\rightarrow$ **bool**

prime(x) $=_{\text{def}}$ « x is a prime number ».

Using, additionally,

prod: **psequ** $\rightarrow$ **nat**

prod(sp) $=_{\text{def}}$ **if** sp = [] **then** 1 **else hd**sp * *prod*(**tl**sp) **fi**

and

ordered: **psequ** $\rightarrow$ **bool**

ordered(sp) $=_{\text{def}}$ « sp is increasingly ordered »

a complete formal specification of the problem is given by

all-primes(N) **where**

all-primes: **nat** $\rightarrow$ **set of psequ**

all-primes(n) $=_{\text{def}}$ {sp: **psequ** || *prod*(sp) = n $\wedge$ *ordered*(sp)}.

The result of rule application is

primes(N) **where**

primes: **nat** $\rightarrow$ **set of psequ**

primes(n) $=_{\text{def}}$ *prs*(n, [], 2)

prs: (**nat** $\times$ **psequ** $\times$ **nat**) $\rightarrow$ **set of psequ**

prs(n, s, c) $=_{\text{def}}$

if n = 1 **then** {s}

elsf true then *prs*(**if** c | n **then** n **div** c **else** n **fi**, **if** c | n **then** (s++c, c) **else** (s, *next*(c)) **fi**)

else Ø **fi**

next: **nat** $\rightarrow$ **nat**

next(c) $=_{\text{def}}$ **if** c=2 **then** 3 **else** c+2 **fi**

where *prs* can be further transformed (using simplification of conditional; distributivity of function call and conditional; simplification) into

prs(n, s, c) $=_{\text{def}}$

if n = 1 **then** {s}

elsf c | n **then** *prs*(n **div** c, s++c, c)

else *prs*(n, s, *next*(c)) **fi**

3.5.3 Another application: all primes up to a given natural number

Given a natural number n (≥ 1), it is asked to compute the set of all primes less or equal to n.

Using

prime: **nat** $\rightarrow$ **bool**
prime(x) $=_{\text{def}}$ « x is a prime number »

the problem is obviously formally specified by

all-primes(N) **where**

all-primes: **nat** $\rightarrow$ **set of nat**
all-primes(n) $=_{\text{def}}$ $\{p$: **nat** $\| \ 1 \leq p \leq n \wedge prime(p)\}$.

The result of rule application is

primes(N) **where**

primes: **nat** $\rightarrow$ **set of nat**
primes(n) $=_{\text{def}}$ *prs*$(n, 1, \emptyset)$

prs: (**nat** $\times$ **nat** $\times$ **set of nat**) $\rightarrow$ **set of nat**
prs(n, q, s) $=_{\text{def}}$
if $q = n$ **then** s
elsf true then *prs*$(n$, **if** *prime*(q) **then** $(q+1, s \cup \{q\})$ **else** $(q+1, s)$ **fi**)
else $\emptyset$ **fi**

where *prs* can be further transformed (using simplification of conditional; distributivity of function call and conditional; simplification) into

prs(n, q, s) $=_{\text{def}}$
if $q = n$ **then** s
elsf *prime*(q) **then** *prs*$(n, q+1, s \cup \{q\})$
else *prs*$(n, q+1, s)$ **fi**

3.6 Further variants

Similar to variants 1 to 3 (where the initial specification asked for some expression over a set rather than the set itself), further variants could be given (and proved in an analogous way) to deal with other set expressions (e.g., **min**, **max**, or the like).

So far, we also have only considered variants where the alternatives resulting from the splitting of the computation according to applicability condition (4) are considered in an ordered way. Of course, we can also give analogous variants where the union

$$\bigcup_{i = 1..n(u,v)} \{r(X, y) \parallel \mathrm{B}(i, u, v) \,\Delta\, \mathrm{Q}'(\mathrm{K}(i, u, v), \mathrm{R}(i, u, v), y)\}$$

in condition 4 is replaced by

$$\bigcup_{x \in m(u,v)} \{r(X, y) \parallel \mathrm{B}(x, u, v) \,\Delta\, \mathrm{Q}'(\mathrm{K}(x, u, v), \mathrm{R}(x, u, v), y)\}$$

and by using

$m(u, v) \setminus \{x\}$	for	$next(k, u,v)$
$u \setminus \{x\}$	for	$\mathrm{K}(x, u, v)$
$m(u, v) = \emptyset$	for	$k > n(u, v)$.

Since, however, efficient computation requires an ordering of these alternatives anyhow, we have refrained from giving these variants explicitly.

4. Related work and concluding remarks

As its main contribution, this paper has presented a powerful transformation rule to convert specifications of set-valued functions defined by set comprehension into executable functional implementations that basically show the behaviour of backtrack algorithms. From this central transformation rule we have furthermore derived several variants to cope with related problems and their solution by backtracking.

4.1 Related work

Early papers that dealt with the problem of backtracking from a methodical point of view are [Gerhart, Yelowitz 76] and [Broy, Wirsing 80]. Both papers give kind of functional program schemes to characterize and discuss the idea of backtracking, but do not investigate the problem of how to transform descriptive specifications into these functional schemes.

A treatment of backtracking as a solution to search problems which is fairly close to our approach can be found in the work of Doug Smith (cf., e.g., [Smith 86], [Smith 88], or [Smith 90] – also for further references). He gives various design strategies which achieve roughly the same effect as the application of our rule. Also the entirety of conditions required to apply these strategies is basically comparable to our applicability conditions. The main difference in the two approaches – apart from having a strategy there and a compact rule here – is in various technical aspects. Also, it is not quite clear how the variants we deal with fit into Smith's approach.

Of course, our rules presented in this paper also show many relationships to the rules presented in [Partsch 90]. E.g., the rule "∃-elimination by backtracking II" may be considered a specialized form of variant 2 presented here, and the rule "computing qualifiers for a finite set" is but a particular special case of our central rule. Moreover, a difference in methodically looking at the problem is worth mentioned here. Whereas in [Partsch 90] we primarily concentrate on solutions for the existence problem and present tactics to extend these solutions to also cover the "some" or the "set-of-all" problem, we started here the other way round which considerably reduces the amount of work for concrete applications.

4.2 Concluding remarks

In some sense, the problems our rule is able to deal with, can be viewed as generate-and-test problems to produce sets the elements of which are characterized by some predicate. Although the only real restriction for the application of the rule is provided by the applicability conditions, best use of the rule can be made for those problems where the elements of the solution are "complex" (i.e., not directly computable from the input) and where the search space is large (maybe infinite) and not easily enumerable. For "simpler" problems (not showing these characteristics) application of the rule is well possible, but usually will require additional effort to simplify the result.

It is obvious, that our rule is primarily aimed at languages with the initially mentioned semantic characteristics. For these, the rule offers a possibility to convert non-operational, descriptive specifications into operational ones – maybe as a first step towards further improving efficiency or switching to an imperative implementation. In the context of a non-strict, lazy functional language (such as Gofer), the value of our rule may be doubted, since in such a language our initial specifications may be directly executed. For all our examples, however, corresponding comparisons (with Gofer implementations of the respective program versions) have shown a substantial increase in efficiency (through application of our rule) also for a non-strict, lazy functional language.

The main problem in using the given transformation rules for concrete applications is in finding the appropriate instantiations required by the applicability conditions. Although some guidance (in finding instantiations) is given by the conditions themselves as well as by the remarks in section 1, three major creative activities remain to be done:

- finding a suitable way of splitting the problem into subproblems (condition (4))
- inventing an appropriate invariant P (conditions (2) and (5)), and
- defining a suitable ordering for the proof of termination (condition (6)).

Once a suitable instantiation has been found, the verification of the resulting proof obligations is straightforward and even gives potential for automation using a theorem prover. Experience (when dealing with all the examples given here and in [Achatz, Partsch 96]) has confirmed the assumption that the reverse of this statement also holds: Whenever complicated proof obligations resulted from some instantiation, it turned out that the real problem was with the instantiation rather than the proof obligation.

References

~~[Achatz, Partsch 96]~~

~~Achatz, K., Partsch, H.: A powerful transformation rule, its applications and variants. Faculty for Informatics, University of Ulm, Technical Report 1996 (to appear)~~

[Bauer et al. 85]Bauer, F.L., Berghammer, R., Broy, M., Dosch, W., Geiselbrechtinger, F., Gnatz, R., Hangel, E., Hesse, W., Krieg-Brückner, B., Laut, A., Matzner, T., Möller, B., Nickl, F., Partsch, H., Pepper, P., Samelson, K., Wirsing, M., Wössner, H.: The Munich project CIP. Volume I: The wide spectrum language CIP-L. Lecture Notes in Computer Science **183**, Berlin: Springer 1985

[Bauer et al. 89]

Bauer, F.L., Möller, B., Partsch, H., Pepper, P.: Programming by formal reasoning - computer-aided intuition-guided programming. IEEE Transactions on Software Engineering **15**:2, 165-180 (1989)

[Boiten et al. 92]

Boiten, E.A., Partsch, H.A., Tuijnman, D., Völker, N.: How to produce correct software - an introduction to formal specification and program development by transformations. The Computer Journal **35**:6, 547-554 (1992)

[Broy, Wirsing 90]

Broy, M., Wirsing, M.: Program development: from enumeration to backtracking. Information Processing letters **10**:4,5, 193-197 (1980)

[Gerhart, Yelowitz 76]

Gerhart, S.L., Yelowitz, L.: Control structure abstractions of the backtracking programming technique. Proc. 2nd Int. Conf. on Software Engineering, October 13-15, 1976, San Francisco, Ca., pp. 44-49. Also: IEEE Transactions on Software Engineering **2**:4, 285-292 (1976)

[Partsch 90]

Partsch, H.A.: Specification and transformation of programs - a formal approach to software development. Berlin: Springer 1990

[Smith 86]

Smith, D.R.: On the design of generate-and-test algorithms: subspace generators. Technical Report KES.U.86.6, Kestrel Institute, Palo Alto, Ca., 1986. Also: Meertens, L.G.L.T. (ed.): Program specification and transformation. Amsterdam: North-Holland 1987, pp. 207-220.

[Smith 88]

Smith, D.R.: The structure and design of global search algorithms.Technical Report KES.U.87.12, Kestrel Institute, Palo Alto, Ca., 1988

[Smith 90]

Smith, D.R.: KIDS: a semiautomatic program development system. IEEE Transactions on Software Engineering **16**:9, 1024-1043 (1990)

Appendix

A.1 Permutations (section 2.2)

Instantiations

f	**by**	*perms*
g	**by**	p
g'	**by**	p'
g''	**by**	p''
X	**by**	Q
x	**by**	q
y	**by**	sp
u	**by**	q
v	**by**	qp
$r(X, y)$	**by**	sp
$\mathrm{Q}(x, y)$	**by**	$same\text{-}els(q, sp)$
$\mathrm{P}(X, u, v)$	**by**	$same\text{-}els(Q, q{+}{+}qp)$
E	**by**	$[]$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\, s'$: **ndsequ** $\Vert\ sp = qp{+}{+}s' \wedge same\text{-}els(q, s')$
$\mathrm{T}(u, v)$	**by**	$q = []$
$\mathrm{H}(u, v)$	**by**	$\{qp\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	$\lvert q \rvert$
$\mathrm{B}(i, u, v)$	**by**	**true**
$\mathrm{K}(i, u, v)$	**by**	$q\text{-}q[k]$
$\mathrm{R}(i, u, v)$	**by**	$qp{+}{+}q[k]$
$next(k, u, v)$	**by**	$k+1$
WF-ORD(**m**, <)	**by**	(**ndsequ**, $a < b \Leftrightarrow \lvert a \rvert < \lvert b \rvert$).

Proof obligations

(0) $(\lvert\{sp$: **ndsequ** $\Vert\ same\text{-}els(q, sp)\}\rvert < \infty) \equiv$ **true**

(1) $same\text{-}els(Q, Q{+}{+}[]) \equiv$ **true**

(2) $same\text{-}els(q, sp) \equiv \exists\, s'$: **ndsequ** $\Vert\ sp = []{+}{+}s' \wedge same\text{-}els(q, s')$

(3) $same\text{-}els(Q, q{+}{+}qp)\ \Delta\ q = [] \equiv$ **true** $\vdash$
$\{qp\} \equiv \{sp$: **ndsequ** $\Vert\ \exists\, s'$: **ndsequ** $\Vert\ sp = qp{+}{+}s' \wedge same\text{-}els(q, s')\}$

(4) $same\text{-}els(Q, q{+}{+}qp)\ \Delta\ q \neq [\,] \equiv \mathbf{true} \vdash$

$\{sp\text{: } \mathbf{ndsequ} \parallel \exists\ s'\text{: } \mathbf{ndsequ} \parallel sp = qp{+}{+}s' \ \wedge\ same\text{-}els(q, s')\} \equiv$

$\bigcup_{i\,=\,1..|q|}\ \{sp\text{: } \mathbf{ndsequ} \parallel \exists\ s'\text{: } \mathbf{ndsequ} \parallel sp = (qp{+}{+}q[i]){+}{+}s' \ \wedge\ same\text{-}els(q{-}q[i], s')\}$

(5) $1 \leq i \leq |q|\ \Delta\ same\text{-}els(Q, q{+}{+}qp)\ \Delta\ q \neq [\,] \equiv \mathbf{true} \vdash$

$same\text{-}els(Q, (q{-}q[i]){+}{+}(qp{+}{+}q[i])) \equiv \mathbf{true}$

(6) $1 \leq i \leq |q| \ \equiv \mathbf{true} \vdash |q - q[i]| < |q| \equiv \mathbf{true}$

A.2 Segmentations (section 2.3)

Instantiations

f	**by**	*all-segs*
g	**by**	*segs*
g'	**by**	*segs'*
g''	**by**	*segs''*
X	**by**	Q
x	**by**	q
y	**by**	sp
u	**by**	q
v	**by**	qp
$r(X, y)$	**by**	sp
$\mathrm{Q}(x, y)$	**by**	$flatten(sp) = q$
$\mathrm{P}(X, u, v)$	**by**	$flatten(qp{+}{+}q) = Q$
E	**by**	$[\,]$
$\mathrm{Q'}(u, v, y)$	**by**	$\exists\ s'\text{: } \mathbf{ssequ} \parallel flatten(s') = q\ \wedge$ $(sp = qp{+}{+}s' \ \vee (s' \neq [\,]\ \Delta sp = qp{+}{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))$
$\mathrm{T}(u, v)$	**by**	$q = [\,]$
$\mathrm{H}(u, v)$	**by**	$\{qp\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	2
$\mathrm{B}(i, u, v)$	**by**	**true**
$\mathrm{K}(i, u, v)$	**by**	$\mathbf{tl}q$
$\mathrm{R}(i, u, v)$	**by**	**if** $k{=}1$ **then** $qp{+}{+}_{\mathrm{tl}}[\mathbf{hd}q]$ **else** $qp{+}{+}[[\mathbf{hd}q]]$ **fi**
$next(k, u, v)$	**by**	$k{+}1$
WF-ORD($\mathbf{m}$, <)	**by**	$(\mathbf{msequ},\ a < b \ \Leftrightarrow |a| < |b|)$.

Proof obligations

(0) $(|\{sp\text{: } \mathbf{ssequ} \parallel flatten(sp) = q\}| < \infty) \equiv \mathbf{true}$

(1) $flatten([]{+}{+}Q) = Q \equiv \mathbf{true}$

(2) $flatten(sp) = q \equiv \exists\, s'\text{: }\mathbf{sseq u} \,\|\, flatten(s') = q \wedge (sp = []{+}{+}s' \vee (s' \neq [] \,\Delta\, sp = []{+}{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))$

(3) $flatten(qp{+}{+}q) = Q \,\Delta\, q = [] \equiv \mathbf{true} \vdash$
$\{qp\} \equiv \{sp\text{: }\mathbf{sseq u} \,\|\, \exists\, s'\text{: }\mathbf{sseq u} \,\|\, flatten(s') = q \wedge$
$(sp = qp{+}{+}s' \vee (s' \neq [] \,\Delta\, sp = qp{+}{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))\}$

(4) $flatten(qp{+}{+}q) = Q \,\Delta\, q \neq [] \equiv \mathbf{true} \vdash$
$\{sp\text{: }\mathbf{sseq u} \,\|\, \exists\, s'\text{: }\mathbf{sseq u} \,\|\, flatten(s') = q \wedge (sp = qp{+}{+}s' \vee (s' \neq [] \,\Delta\, sp = qp{+}{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))\} \equiv$
$\bigcup_{i = 1..2} \{sp\text{: }\mathbf{sseq u} \,\|\, \exists\, s'\text{: }\mathbf{sseq u} \,\|\, flatten(s') = \mathbf{tl}q \wedge$
$(sp = \mathbf{if}\ k{=}1\ \mathbf{then}\ qp{+}{+}_{\mathrm{tl}}[\mathbf{hd}q]\ \mathbf{else}\ qp{+}{+}[[\mathbf{hd}q]]\ \mathbf{fi}{+}{+}s' \vee$
$sp = \mathbf{if}\ k{=}1\ \mathbf{then}\ qp{+}{+}_{\mathrm{tl}}[\mathbf{hd}q]\ \mathbf{else}\ qp{+}{+}[[\mathbf{hd}q]]\ \mathbf{fi}{+}{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s')\}$

(5) $1 \leq i \leq 2 \,\Delta\, flatten(qp{+}{+}q) = Q \,\Delta\, q \neq [] \equiv \mathbf{true} \vdash$
$flatten(\mathbf{if}\ k{=}1\ \mathbf{then}\ qp{+}{+}_{\mathrm{tl}}[\mathbf{hd}q]\ \mathbf{else}\ qp{+}{+}[[\mathbf{hd}q]]\ \mathbf{fi}\ {+}{+}\mathbf{tl}q) = Q) \equiv \mathbf{true}$

(6) $1 \leq i \leq 2 \equiv \mathbf{true} \vdash |\mathbf{tl}q| < |q| \equiv \mathbf{true}$

A.3 Partitions of a natural number (section 2.4)

Instantiations

f	**by**	*all-parts*
g	**by**	*parts*
g'	**by**	*parts'*
g''	**by**	*parts''*
X	**by**	Q
x	**by**	q
y	**by**	sp
u	**by**	q
v	**by**	qp
$r(X, y)$	**by**	sp
$\mathrm{Q}(x, y)$	**by**	$sum(sp) = q$
$\mathrm{P}(X, u, v)$	**by**	$sum(qp)+q = Q$
E	**by**	[1]
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\, s'\text{: }\mathbf{psequ} \,\|\, sum(s') = q \wedge$ $(sp = qp{+}{+}s' \vee (s' \neq [] \,\Delta\, sp = qp{+}_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))$
$\mathrm{T}(u, v)$	**by**	$q = 0$
$\mathrm{H}(u, v)$	**by**	$\{qp\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$

$n(u, v)$	**by**	2
$\mathrm{B}(i, u, v)$	**by**	**true**
$\mathrm{K}(i, u, v)$	**by**	$q-1$
$\mathrm{R}(i, u, v)$	**by**	**if** $k=1$ **then** $qp+_{\mathrm{tl}}1$ **else** qp++[1] **fi**
$next(k, u, v)$	**by**	$k+1$
WF-ORD(**m**, <)	**by**	(**nat**, <).

Proof obligations

(0) $(|\{sp\text{: } \mathbf{psequ} \parallel sum(sp) = q\}| < \infty) \equiv \mathbf{true}$

(1) $sum([])+Q = Q \equiv \mathbf{true}$

(2) $sum(sp) = q \equiv \exists\ s'\text{: } \mathbf{psequ} \parallel sum(s') = q \wedge (sp = [1]{+}{+}s' \ \vee (s' \neq [] \ \Delta\ sp = [1]+_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))$

(3) $sum(qp)+q = Q \ \Delta\ q = 0 \ \equiv \mathbf{true} \vdash$
$\{qp\} \equiv \{sp\text{: } \mathbf{psequ} \parallel \exists\ s'\text{: } \mathbf{psequ} \parallel sum(s') = q \wedge (sp = qp{+}{+}s' \ \vee (s' \neq [] \ \Delta\ sp = qp+_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))\}$

(4) $sum(qp)+q = Q \ \Delta\ q \neq 0 \ \equiv \mathbf{true} \vdash$
$\{sp\text{: } \mathbf{psequ} \parallel \exists\ s'\text{: } \mathbf{psequ} \parallel sum(s') = q \wedge (sp = qp{+}{+}s' \ \vee (s' \neq [] \ \Delta\ sp = qp+_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s'))\} \equiv$
$\bigcup_{i = 1..2}\ \{sp\text{: } \mathbf{psequ} \parallel \exists\ s'\text{: } \mathbf{psequ} \parallel sum(s') = q-1 \ \wedge$
$(sp = \mathbf{if}\ k{=}1\ \mathbf{then}\ qp+_{\mathrm{tl}}1\ \mathbf{else}\ qp{+}{+}[1]\ \mathbf{fi}{+}{+}s' \ \vee$
$sp = \mathbf{if}\ k{=}1\ \mathbf{then}\ qp+_{\mathrm{tl}}1\ \mathbf{else}\ qp{+}{+}[1]\ \mathbf{fi}+_{\mathrm{lst}}\mathbf{hd}s'{+}{+}\mathbf{tl}s')\}$

(5) $1 \leq i \leq 2 \ \Delta\ sum(qp)+q = Q \ \Delta\ q \neq 0 \equiv \mathbf{true} \vdash$
$sum(\mathbf{if}\ k{=}1\ \mathbf{then}\ qp+_{\mathrm{tl}}1\ \mathbf{else}\ qp{+}{+}[1]\ \mathbf{fi})+(q-1) = Q \equiv \mathbf{true}$

(6) $1 \leq i \leq 2 \ \equiv \mathbf{true} \vdash q-1 < q \equiv \mathbf{true}$

A.4 All subsequences (section 2.4)

Instantiations

f	**by**	*all-subs*
g	**by**	*subs*
g'	**by**	*subs'*
g''	**by**	*subs"*
X	**by**	Q
x	**by**	q
y	**by**	q'
u	**by**	q
v	**by**	p
$r(X, y)$	**by**	q'

$Q(x, y)$	**by**	$issubssequ(q', q)$
$P(X, u, v)$	**by**	$\exists\, p'$: **msequ** $\Vert\ issubsequ(p, p') \wedge p'{+}{+}q = Q$
E	**by**	$[]$
$Q'(u, v, y)$	**by**	$\exists\, s'$: **msequ** $\Vert\ q' = p{+}{+}s' \wedge issubssequ(s', q)$
$T(u, v)$	**by**	$q = []$
$H(u, v)$	**by**	$\{p\}$
$D(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	2
$B(i, u, v)$	**by**	**true**
$K(i, u, v)$	**by**	**tl**q
$R(i, u, v)$	**by**	**if** k=1 **then** p **else** p++[**hd**q] **fi**
$next(k, u, v)$	**by**	$k+1$
WF-ORD(**m**, <)	**by**	(**msequ**, $a < b \Leftrightarrow \vert a\vert < \vert b\vert$).

Proof obligations

(0) $(\vert\{q'$: **msequ** $\Vert\ issubssequ(q', q)\}\vert < \infty) \equiv$ **true**

(1) $\exists\, p'$: **msequ** $\Vert\ issubsequ([], p') \wedge p'{+}{+}Q = Q \equiv$ **true**

(2) $issubssequ(q', q) \equiv \exists\, s'$: **msequ** $\Vert\ q' = []{+}{+}s' \wedge issubssequ(s', q)$

(3) $(\exists\, p'$: **msequ** $\Vert\ issubsequ(p, p') \wedge p'{+}{+}q = Q)\ \Delta\ q = []\ \equiv$ **true** $\vdash$
$\{p\} \equiv \{q'$: **msequ** $\Vert\ \exists\, s'$: **msequ** $\Vert\ q' = p{+}{+}s' \wedge issubssequ(s', q)\}$

(4) $(\exists\, p'$: **msequ** $\Vert\ issubsequ(p, p') \wedge p'{+}{+}q = Q)\ \Delta\ q \neq []\ \equiv$ **true** $\vdash$
$\{q'$: **msequ** $\Vert\ \exists\, s'$: **msequ** $\Vert\ q' = p{+}{+}s' \wedge issubssequ(s', q)\} \equiv$
$\bigcup_{i = 1..2}\ \{q'$: **ssequ** $\Vert\ \exists\, s'$: **msequ** $\Vert\ q' =$ **if** k=1 **then** p **else** p++[**hd**q] **fi**++$s' \wedge$
$issubssequ(s',$ **tl**$q)\}$

(5) $1 \le i \le 2\ \Delta\ (\exists\, p'$: **msequ** $\Vert\ issubsequ(p, p') \wedge p'{+}{+}q = Q)\ \Delta\ q \neq [] \equiv$ **true** $\vdash$
$(\exists\, p'$: **msequ** $\Vert\ issubsequ($**if** k=1 **then** p **else** p++[**hd**q] **fi**, $p') \wedge p'$++**tl**$q = Q) \equiv$ **true**

(6) $1 \le i \le 2\ \equiv$ **true** $\vdash \vert$**tl**$q\vert < \vert q\vert \equiv$ **true**

A.5 The Pack-Problem (section 2.6)

Instantiations

f	**by**	*all-packs*
g	**by**	*ap*
g'	**by**	*ap'*
g''	**by**	*ap''*

X	**by**	(K, B)
x	**by**	(k, b)
y	**by**	A
u	**by**	(k, b)
v	**by**	a
$r(X, y)$	**by**	A
$\mathrm{Q}(x, y)$	**by**	$legal(A, k, b)$
$\mathrm{P}(X, u, v)$	**by**	$legal(a, dom(a), B) \wedge dom(a){+}{+}k = K$
E	**by**	$\emptyset$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\, a'\!: \mathbf{corr} \parallel A = a \circ a' \wedge legal(a', k, b)$
$\mathrm{T}(u, v)$	**by**	$k = []$
$\mathrm{H}(u, v)$	**by**	$\{a\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	$\lvert b \rvert$
$\mathrm{B}(i, u, v)$	**by**	$fits(\mathbf{hd}k, b[i])$
$\mathrm{K}(i, u, v)$	**by**	$(\mathbf{tl}k, b[i \,\Theta\, \lvert \mathbf{hd}k \rvert])$
$\mathrm{R}(i, u, v)$	**by**	$a[\mathbf{hd}k] \leftarrow b[i]$
$next(k, u, v)$	**by**	$k{+}1$
WF-ORD($\mathbf{r}$, <)	**by**	$(\mathbf{blocks}, a < b \iff \lvert a \rvert < \lvert b \rvert)$

Proof obligations

(0) $(\lvert\{A\!: \mathbf{corr} \parallel legal(A, k, b)\}\rvert < \infty) \equiv \mathbf{true}$

(1) $(legal(\emptyset, dom(\emptyset), B) \wedge dom(\emptyset){+}{+}K = K) \equiv \mathbf{true}$

(2) $legal(A, k, b) \equiv \exists\, a'\!: \mathbf{corr} \parallel A = \emptyset \circ a' \wedge legal(a', k, b)$

(3) $(legal(a, dom(a), B) \wedge dom(a){+}{+}k = K) \,\Delta\, k = [] \;\equiv \mathbf{true} \vdash$
$\{a\} \equiv \{A\!: \mathbf{corr} \parallel \exists\, a'\!: \mathbf{corr} \parallel A = a \circ a' \wedge legal(a', k, b)\}$

(4) $(legal(a, dom(a), B) \wedge dom(a){+}{+}k = K) \,\Delta\, k \neq [] \;\equiv \mathbf{true} \vdash$
$\{A\!: \mathbf{corr} \parallel \exists\, a'\!: \mathbf{corr} \parallel A = a \circ a' \wedge legal(a', k, b)\} \equiv$
$\bigcup_{i = 1..n} \{A\!: \mathbf{corr} \parallel fits(\mathbf{hd}k, b[i]) \,\Delta$
$\exists\, a'\!: \mathbf{corr} \parallel A = (a[\mathbf{hd}k] \leftarrow b[i]) \circ a' \wedge legal(a', \mathbf{tl}k, b[i \,\Theta\, \lvert \mathbf{tl}k \rvert])\}$

(5) $1 \leq i \leq \lvert b \rvert \;\Delta\, (legal(a, dom(a), B) \wedge dom(a){+}{+}k = K) \,\Delta\, k \neq [] \,\Delta\, fits(\mathbf{hd}k, b[i]) \equiv \mathbf{true} \vdash$
$(legal(a[\mathbf{hd}k] \leftarrow b[i], dom(a[\mathbf{hd}k] \leftarrow b[i]), B) \wedge dom(a[\mathbf{hd}k] \leftarrow b[i]){+}{+}\mathbf{tl}k = K) \equiv \mathbf{true}$

(6) $1 \leq i \leq \lvert b \rvert \;\Delta\, fits(\mathbf{hd}k, b[i]) \;\equiv \mathbf{true} \vdash \lvert \mathbf{tl}k \rvert \;< \lvert k \rvert \equiv \mathbf{true}$

A.6 Topological Sortings (section 2.7)

Instantiations

f	**by**	*top-sorts*
g	**by**	*ts*
g'	**by**	*ts'*
g''	**by**	*ts''*
X	**by**	(M, PO)
x	**by**	(m, po)
y	**by**	t
u	**by**	(m, po)
v	**by**	to
$r(X, y)$	**by**	t
$\mathrm{Q}(x, y)$	**by**	$\mathit{same\text{-}els}(t, m) \wedge \mathit{respects}(t, po)$
$\mathrm{P}(X, u, v)$	**by**	$\mathit{same\text{-}els}(M, to{+}{+}m)$
E	**by**	$[]$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\, s'$: **ndsequ** $\Vert\; t = to{+}{+}s' \;\wedge \mathit{same\text{-}els}(s', m) \wedge \mathit{respects}(s', po)$
$\mathrm{T}(u, v)$	**by**	$m = []$
$\mathrm{H}(u, v)$	**by**	$\{to\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	$\lvert m \rvert$
$\mathrm{B}(i, u, v)$	**by**	$\mathit{ismin}(m[i], po)$
$\mathrm{K}(i, u, v)$	**by**	$(m\text{-}m[i], \mathit{rem}(m[i], po))$
$\mathrm{R}(i, u, v)$	**by**	$to{+}{+}m[i]$
$\mathit{next}(k, u, v)$	**by**	$k+1$
WF-ORD(**m**, <)	**by**	(**ndsequ**, $a < b \;\Leftrightarrow \lvert a \rvert < \lvert b \rvert$)

Proof obligations

(0) $(\lvert\{t$: **ndsequ** $\Vert\; \mathit{same\text{-}els}(t, m) \wedge \mathit{respects}(t, po)\}\rvert < \infty) \equiv$ **true**

(1) $\mathit{same\text{-}els}(M, []{+}{+}M) \equiv$ **true**

(2) $\mathit{same\text{-}els}(t, m) \wedge \mathit{respects}(t, po) \equiv$
$\exists\, s'$: **ndsequ** $\Vert\; t = []{+}{+}s' \;\wedge \mathit{same\text{-}els}(s', m) \wedge \mathit{respects}(s', po)$

(3) $\mathit{same\text{-}els}(M, to{+}{+}m)\; \Delta\; m = [] \;\equiv$ **true** $\vdash$
$\{to\} \equiv \{t$: **ndsequ** $\Vert\; \exists\, s'$: **ndsequ** $\Vert\; t = to{+}{+}s' \;\wedge \mathit{same\text{-}els}(s', m) \wedge \mathit{respects}(s', po)\}$

(4) $same\text{-}els(M, to{+}{+}m)\ \Delta\ m \neq [\,] \equiv \mathbf{true} \vdash$
$\{t\text{: } \mathbf{ndsequ} \parallel \exists\, s'\text{: } \mathbf{ndsequ} \parallel t = to{+}{+}s' \wedge same\text{-}els(s', m) \wedge respects(s', po)\} \equiv$
$\bigcup_{i = 1..|m|} \{t\text{: } \mathbf{ndsequ} \parallel ismin(m[i], po)\ \Delta\ \exists\, s'\text{: } \mathbf{ndsequ} \parallel t = to{+}{+}m[i]{+}{+}s' \wedge$
$same\text{-}els(s', m\text{-}m[i]) \wedge respects(s', rem(m[k], po))\}$

(5) $1 \leq i \leq |m|\ \Delta\ same\text{-}els(M, to{+}{+}m)\ \Delta\ m \neq [\,]\ \Delta\ ismin(m[i], po) \equiv \mathbf{true} \vdash$
$same\text{-}els(M, (to{+}{+}m[i]){+}{+}(m{-}m[i])) \equiv \mathbf{true}$

(6) $1 \leq i \leq |m|\ \Delta\ ismin(m[i], po) \equiv \mathbf{true} \vdash |m - m[i]| < |m| \equiv \mathbf{true}$

A.7 Maximal paths in a graph (section 2.8)

Instantiations

f	**by**	*all-paths*
g	**by**	*paths*
g'	**by**	*paths'*
g''	**by**	*paths''*
X	**by**	(N, G)
x	**by**	(n, g)
y	**by**	t
u	**by**	(n, g)
v	**by**	ns
$r(X, y)$	**by**	t
$Q(x, y)$	**by**	$ismaximal(t, n, g)$
$P(X, u, v)$	**by**	$\mathbf{hd}ns = n \wedge connected(ns, g) \wedge acyclic(ns)$
E	**by**	$[n]$
$Q'(u, v, y)$	**by**	$\exists\, s'\text{: } \mathbf{nsequ} \parallel t = ns{+}{+}s' \wedge ismaximal(t, n, g)$
$T(u, v)$	**by**	$isend(ns, g)$
$H(u, v)$	**by**	$\{ns\}$
$D(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	$\lvert succs(\mathbf{lst}ns, g)\rvert$
$B(i, u, v)$	**by**	$succs(\mathbf{lst}ns, g)[i] \notin ns$
$K(i, u, v)$	**by**	(n, g)
$R(i, u, v)$	**by**	$ns{+}{+}succs(\mathbf{lst}ns, g)[i]$
$next(k, u, v)$	**by**	$k{+}1$
WF-ORD($\mathbf{m}$, <)	**by**	$(\mathbf{nsequ},\ a < b \Leftrightarrow nodes(g){-}|a| < nodes(g){-}|b|)$

Proof obligations

(0) $(|\{t\text{: } \mathbf{nsequ} \parallel ismaximal(t, n, g)\}| < \infty) \equiv \mathbf{true}$

(1) $\mathbf{hd}[N] = N \wedge connected([N], G) \wedge acyclic([N]) \equiv \mathbf{true}$

(2) $ismaximal(t, n, g) \equiv \exists\, s'\text{: } \mathbf{nsequ} \parallel t = [n]{+}{+}s' \;\wedge ismaximal(t, n, g)$

(3) $(\mathbf{hd}ns = n \wedge connected(ns, g) \wedge acyclic(ns))\ \Delta\ isend(ns, g) \equiv \mathbf{true} \vdash$
$\{ns\} \equiv \{t\text{: } \mathbf{nsequ} \parallel \exists\, s'\text{: } \mathbf{nsequ} \parallel t = ns{+}{+}s' \;\wedge ismaximal(t, n, g)\}$

(4) $(\mathbf{hd}ns = n \wedge connected(ns, g) \wedge acyclic(ns))\ \Delta\ \neg isend(ns, g) \equiv \mathbf{true} \vdash$
$\{t\text{: } \mathbf{nsequ} \parallel \exists\, s'\text{: } \mathbf{nsequ} \parallel t = ns{+}{+}s' \;\wedge ismaximal(t, n, g)\} \equiv$
$\bigcup_{i = 1..|m|} \{t\text{: } \mathbf{nsequ} \parallel succs(\mathbf{lst}ns, g)[i] \notin ns\ \Delta$
$\exists\, s'\text{: } \mathbf{nsequ} \parallel t = ns{+}{+}succs(\mathbf{lst}ns, g)[i]{+}{+}s' \;\wedge ismaximal(t, n, g)\}$

(5) $1 \leq i \leq |succs(\mathbf{lst}ns, g)|\ \Delta\ (\mathbf{hd}ns = n \wedge connected(ns, g) \wedge acyclic(ns))\ \Delta$
$\neg isend(ns, g)\ \Delta\ succs(\mathbf{lst}ns, g)[k] \notin ns \equiv \mathbf{true} \vdash$
$\mathbf{hd}ns = n \wedge connected(ns{+}{+}succs(\mathbf{lst}ns, g)[i], g) \wedge acyclic(ns{+}{+}succs(\mathbf{lst}ns, g)[i]) \equiv \mathbf{true}$

(6) $1 \leq i \leq |succs(\mathbf{lst}ns, g)|\ \Delta\ succs(\mathbf{lst}ns, g)[k] \notin ns \equiv \mathbf{true} \vdash$
$nodes(g) - |ns{+}{+}succs(\mathbf{lst}ns, g)[i]| < nodes(g) - |ns| \equiv \mathbf{true}$

A.8 A Prolog Interpreter (section 2.9)

Instantiations

f	**by**	*interprete*
g	**by**	*int*
g'	**by**	*int'*
g''	**by**	*int''*
X	**by**	(P, A)
x	**by**	(p, gl)
y	**by**	Θ
u	**by**	(p, gl)
v	**by**	Φ
$r(X, y)$	**by**	$\Theta\,|_{vars(A)}$
$\mathrm{Q}(x, y)$	**by**	$(gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta)$
$\mathrm{P}(X, u, v)$	**by**	$(A, \emptyset) \xrightarrow[p]{}{}^{*} (gl, \Phi)$
E	**by**	$\emptyset$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\, \Theta'\text{: } \mathbf{subst} \parallel \Theta = \Phi \circ \Theta' \wedge (gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta')$
$\mathrm{T}(u, v)$	**by**	$gl = []$
$\mathrm{H}(u, v)$	**by**	$\{\Phi\,|_{vars(A)}\}$
$\mathrm{D}(u, v)$	**by**	$\emptyset$
$n(u, v)$	**by**	$|p|$

$B(i, u, v)$	**by**	$\Phi' \neq \mathbf{fail}$ **where** $\Phi' = \mathit{unify}(\mathbf{hd}gl, \mathit{lhs}(\mathit{rn}(p[i], gl)))$
$K(i, u, v)$	**by**	$(\mathit{rhs}(\mathit{rn}(p[i], gl)) ++ \mathbf{tl}gl)\Phi'$
$R(i, u, v)$	**by**	$\Phi \circ \Phi'$

Proof obligations

(0) $(|\{\Theta |_{vars(A)} \parallel (gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta)\}| < \infty) \equiv \mathbf{true}$

(1) $((A, \emptyset) \xrightarrow[p]{}{}^{*} (A, \emptyset)) \equiv \mathbf{true}$

(2) $(gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta) \equiv \exists\, \Theta' \text{: } \mathbf{subst} \parallel \Theta = \emptyset\Theta' \wedge (gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta')$

(3) $(A, \emptyset) \xrightarrow[p]{}{}^{*} (gl, \Phi)\ \Delta\ gl = []\ \equiv \mathbf{true} \vdash$
$\{\Phi |_{vars(A)}\} \equiv \{\Theta |_{vars(A)} \parallel \exists\, \Theta' \text{: } \mathbf{subst} \parallel \Theta = \Phi \circ \Theta' \wedge (gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta')\}$

(4) $(A, \emptyset) \xrightarrow[p]{}{}^{*} (gl, \Phi)\ \Delta\ gl \neq []\ \equiv \mathbf{true} \vdash$
$\{\Theta |_{vars(A)} \parallel \exists\, \Theta' \text{: } \mathbf{subst} \parallel \Theta = \Phi \circ \Theta' \wedge (gl, \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta')\} \equiv$
$\bigcup_{i = 1..|p|} \{\Theta |_{vars(A)} \parallel \Phi' \neq \mathbf{fail}\ \Delta$
$\exists\, \Theta' \text{: } \mathbf{subst} \parallel \Theta = \Phi \circ \Phi' \circ \Theta' \wedge ((\mathit{rhs}(\mathit{rn}(p[i], gl)) ++ \mathbf{tl}gl)\Phi', \emptyset) \xrightarrow[p]{}{}^{*} ([], \Theta')\}$
where $\Phi' = \mathit{unify}(\mathbf{hd}gl, \mathit{lhs}(\mathit{rn}(p[ii], gl)))$

(5) $1 \leq i \leq |p|\ \Delta\ (A, \emptyset) \xrightarrow[p]{}{}^{*} (gl, \Phi)\ \Delta\ gl \neq []\ \Delta\ \Phi' \neq \mathbf{fail}$
where $\Phi' = \mathit{unify}(\mathbf{hd}gl, \mathit{lhs}(\mathit{rn}(p[i], gl))) \equiv \mathbf{true} \vdash$
$(A, \emptyset) \xrightarrow[p]{}{}^{*} ((\mathit{rhs}(\mathit{rn}(p[i], gl)) ++ \mathbf{tl}gl)\Phi', \Phi \circ \Phi') \equiv \mathbf{true}$

A.9 The Coding Problem (section 3.1.2)

Instantiations

f	**by**	*encodings*
g	**by**	*enc*
g'	**by**	*enc'*
g''	**by**	*enc''*
X	**by**	(N, P)
x	**by**	(n, p)
y	**by**	a
z	**by**	a
u	**by**	(n, p)
v	**by**	e
$r(X, y)$	**by**	$\mathit{encode}(a, P)$
$Q(x, y)$	**by**	$\mathit{flatten}(a) = n \wedge \mathit{codable}(a, p)$
$P(X, u, v)$	**by**	$\exists\, z \parallel \mathit{flatten}(z{+}{+}n) = N \wedge e = \mathit{encode}(z, P)$

E	**by**	$\{[]\}$
Q'(u, v, y)	**by**	$\forall e' \in e \parallel \exists a' \parallel a = e'{+}{+}a' \wedge \mathit{flatten}(a') = n \wedge \mathit{codable}(a', p)$
T(u, v)	**by**	$q = []$
H(u, v)	**by**	e
$n(u, v)$	**by**	$\lvert p \rvert$
B(i, u, v)	**by**	$l(p[i]) \,{}^{\cdot}\, n$
K(i, u, v)	**by**	$n \lrcorner l(p[i])$
R(i, u, v)	**by**	$e \oplus r(p[i])$
WF-ORD(**m**, <)	**by**	(**clear**, $a < b \Leftrightarrow \lvert a \rvert < \lvert b \rvert$).

Proof obligations

(0) $(\lvert \bigcup_{a \in deco(n, p)} \mathit{encode}(a, p) \rvert < \infty) \equiv \mathbf{true}$

(1) $\exists z \parallel \mathit{flatten}(z{+}{+}N) = N \wedge \{[]\} = \mathit{encode}(z, P) \equiv \mathbf{true}$

(2) $\mathit{flatten}(a) = n \wedge \mathit{codable}(a, p) \equiv$
$\exists a' \parallel a = []{+}{+}a' \wedge \mathit{flatten}(a') = n \wedge \mathit{codable}(a', p)$

(3) $(\exists z \parallel \mathit{flatten}(z{+}{+}n) = N \wedge e = \mathit{encode}(z, P)) \,\Delta\, n = [] \;\equiv \mathbf{true} \vdash$
$e \equiv \bigcup_{a \in \{a \parallel \forall e' \in e \parallel \exists a' \parallel a = e'{+}{+}a' \wedge \mathit{flatten}(a') = n \wedge \mathit{codable}(a', p)\}} \mathit{encode}(a, P)$

(4) $(\exists z \parallel \mathit{flatten}(z{+}{+}n) = N \wedge e = \mathit{encode}(z, P)) \,\Delta\, n \neq [] \;\equiv \mathbf{true} \vdash$
$\bigcup_{a \in \{a \parallel \forall e' \in e \parallel \exists a' \parallel a = e'{+}{+}a' \wedge \mathit{flatten}(a') = n \wedge \mathit{codable}(a', p)\}} \mathit{encode}(a, P) \equiv$
$\bigcup_{i = 1..\lvert p \rvert} \bigcup_{a \in \{a \parallel l(p[i]) \,{}^{\cdot}\, n \wedge \forall e' \in e \oplus r(p[i]) \parallel \exists a' \parallel a = e'{+}{+}a' \wedge \mathit{flatten}(a') = n \lrcorner l(p[i]) \wedge \mathit{codable}(a', p)\}} \mathit{encode}(a, P)$

(5) $1 \leq i \leq \lvert p \rvert \,\Delta\, (\exists z \parallel \mathit{flatten}(z{+}{+}n) = N \wedge e = \mathit{encode}(z, P)) \,\Delta\, n \neq [] \,\Delta\, l(p[i]) \,{}^{\cdot}\, n \equiv \mathbf{true} \vdash$
$(\exists z \parallel \mathit{flatten}(z{+}{+}n \lrcorner l(p[i])) = N \wedge e \oplus r(p[i]) = \mathit{encode}(z, P)) \equiv \mathbf{true}$

(6) $1 \leq i \leq \lvert p \rvert \;\Delta\, l(p[i]) \,{}^{\cdot}\, n \;\equiv \mathbf{true} \vdash \lvert n \lrcorner l(p[i]) \rvert < \lvert n \rvert \;\equiv \mathbf{true}$

A.10 Top-down recognition (section 3.2.2)

Instantiations

f	**by**	*recognize*
g	**by**	*rec*
g'	**by**	*rec'*
g''	**by**	*rec''*
X	**by**	(P, W)
x	**by**	(p, w)
y	**by**	y
u	**by**	(p, w)

v	**by**	v
Q(x, y)	**by**	$[Z] \underset{p}{\Rightarrow}^{*} y \wedge y = w$
P(X, u, v)	**by**	$\exists w' \parallel w'{+}{+}w = W \wedge [Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}v$
E	**by**	$[Z]$
Q'(u, v, y)	**by**	$v \underset{p}{\Rightarrow}^{*} y \wedge y = w$
T(u, v)	**by**	$w = [] \wedge v = []$
H(u, v)	**by**	**true**
$n(u, v)$	**by**	$\lvert p\rvert+1$
B(i, u, v)	**by**	$(i \le \lvert p\rvert \wedge v \ne [] \,\Delta\, \mathbf{hd}v = lhs(p[i])) \vee$ $(i = \lvert p\rvert+1 \wedge w \ne [] \wedge v \ne [] \,\Delta\, \mathbf{hd}w = \mathbf{hd}v)$
K(i, u, v)	**by**	**if** $i \le \lvert p\rvert$ **then** w **else tl**w **fi**
R(i, u, v)	**by**	**if** $i \le \lvert p\rvert$ **then** $rhs(p[i]){+}{+}\mathbf{tl}v$ **else tl**v **fi**

Proof obligations

(1) $\exists w' \parallel w'{+}{+}W = W \wedge [Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}[Z] \equiv \mathbf{true}$

(2) $[Z] \underset{p}{\Rightarrow}^{*} y \wedge y = w \equiv [Z] \underset{p}{\Rightarrow}^{*} y \wedge y = w$

(3) $(\exists w' \parallel w'{+}{+}w = W \wedge [Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}v) \,\Delta\, (w = [] \wedge v = []) \equiv \mathbf{true} \vdash$
$\mathbf{true} \equiv \exists\, y$: **sequ of term** $\parallel v \underset{p}{\Rightarrow}^{*} y \wedge y = w$

(4) $(\exists w' \parallel w'{+}{+}w = W \wedge [Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}v) \,\Delta\, \neg(w = [] \wedge v = []) \equiv \mathbf{true} \vdash$
$\exists\, y$: **sequ of term** $\parallel v \underset{p}{\Rightarrow}^{*} y \wedge y = w \equiv$
$\exists_{i = 1..\lvert p\rvert+1}\ \exists\, y$: **sequ of term** $\parallel$
$(i \le \lvert p\rvert \wedge v \ne [] \,\Delta\, \mathbf{hd}v = lhs(p[i])) \vee (i = \lvert p\rvert+1 \wedge w \ne [] \wedge v \ne [] \,\Delta\, \mathbf{hd}w = \mathbf{hd}v) \,\Delta$
if $i \le \lvert p\rvert$ **then** $rhs(p[i]){+}{+}\mathbf{tl}v$ **else tl**v **fi** $\underset{p}{\Rightarrow}^{*} y \wedge y =$ **if** $i \le \lvert p\rvert$ **then** w **else tl**w **fi**

(5) $1 \le i \le \lvert p\rvert+1 \,\Delta\, (\exists w' \parallel w'{+}{+}w = W \wedge [Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}v) \,\Delta\, \neg(w = [] \wedge v = []) \,\Delta$
$(i \le \lvert p\rvert \wedge v \ne [] \,\Delta\, \mathbf{hd}v = lhs(p[i])) \vee (i = \lvert p\rvert+1 \wedge w \ne [] \wedge v \ne [] \,\Delta\, \mathbf{hd}w = \mathbf{hd}v) \equiv \mathbf{true} \vdash$
$(\exists w' \parallel w'{+}{+}$**if** $i \le \lvert p\rvert$ **then** w **else tl**w **fi** $= W \wedge$
$[Z] \underset{p}{\Rightarrow}^{*} w'{+}{+}$**if** $i \le \lvert p\rvert$ **then** $rhs(p[i]){+}{+}\mathbf{tl}v$ **else tl**v **fi**$) \equiv \mathbf{true}$

A.11 Shift-Reduce parsing (section 3.3.2)

Instantiations

f	**by**	*parse*
g	**by**	*par*
g'	**by**	*par'*
g''	**by**	*par''*

X	**by**	$(P, [], W)$
x	**by**	(p, st, w)
y	**by**	y
u	**by**	(p, st, w)
v	**by**	sr
$\mathrm{Q}(x, y)$	**by**	$apply(y, st, w) = ([Z], [])$
$\mathrm{P}(X, u, v)$	**by**	$apply(sr, [], W) = (st, w)$
E	**by**	$[]$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists y' \parallel sr{+}{+}y' = y \wedge apply(y', st, w) = ([Z], [])$
$\mathrm{T}(u, v)$	**by**	$w = [] \wedge st = [Z]$
$\mathrm{H}(u, v)$	**by**	sr
$n(u, v)$	**by**	$\lvert p\rvert+1$
$\mathrm{B}(i, u, v)$	**by**	$(i \leq \lvert p\rvert \wedge \exists\, s' \parallel st = s'{+}{+}rhs(p[i])) \vee (i = \lvert p\rvert+1 \wedge w \neq [])$
$\mathrm{K}(i, u, v)$	**by**	**if** $i \leq \lvert p\rvert$ **then** $(s'{+}{+}lhs(p[i]), w)$ **else** $(st{+}{+}\mathbf{hd}w, \mathbf{tl}w)$ **fi**
$\mathrm{R}(i, u, v)$	**by**	**if** $i \leq \lvert p\rvert$ **then** $sr{+}{+}\mathrm{R}(p)$ **else** $sr{+}{+}\mathrm{S}$ **fi**
$next(k, u, v)$	**by**	$k+1$

Proof obligations

(1) $apply([], [], W) = ([], W) \equiv \mathbf{true}$

(2) $apply(y, st, w) = ([Z], []) \equiv \exists y' \parallel []{+}{+}y' = y \wedge apply(y', st, w) = ([Z], [])$

(3) $(apply(sr, [], W) = (st, w))\ \Delta\ (w = [] \wedge st = [Z]) \equiv \mathbf{true} \vdash$
$sr \in \{y\text{: }\mathbf{srsequ} \parallel \exists y' \parallel sr{+}{+}y' = y \wedge apply(y', st, w) = ([Z], [])\} \equiv \mathbf{true}$

(4) $(apply(sr, [], W) = (st, w))\ \Delta\ \neg(w = [] \wedge st = [Z]) \equiv \mathbf{true} \vdash$
some y: **srsequ** $\parallel \exists y' \parallel sr{+}{+}y' = y \wedge apply(y', st, w) = ([Z], []) \equiv$
some y: **srsequ** $\parallel \exists_{i = 1..\lvert p\rvert+1}\ (i \leq \lvert p\rvert \wedge \exists\, s' \parallel st = s'{+}{+}rhs(p[i])) \vee (i = \lvert p\rvert+1 \wedge w \neq [])\ \Delta$
$\exists y' \parallel$ **if** $i \leq \lvert p\rvert$ **then** $sr{+}{+}\mathrm{R}(p)$ **else** $sr{+}{+}\mathrm{S}$ **fi**$++y' = y \wedge$
$apply(y',$ **if** $i \leq \lvert p\rvert$ **then** $(s'{+}{+}lhs(p[i]), w)$ **else** $(st{+}{+}\mathbf{hd}w, \mathbf{tl}w)$ **fi**$) = ([Z], [])$

(5) $1 \leq i \leq \lvert p\rvert+1\ \Delta\ (apply(sr, [], W) = (st, w))\ \Delta\ \neg(w = [] \wedge st = [Z])\ \Delta$
$((i \leq \lvert p\rvert \wedge \exists\, s' \parallel st = s'{+}{+}rhs(p[i])) \vee (i = \lvert p\rvert+1 \wedge w \neq [])) \equiv \mathbf{true} \vdash$
$apply($**if** $i \leq \lvert p\rvert$ **then** $sr{+}{+}\mathrm{R}(p)$ **else** $sr{+}{+}\mathrm{S}$ **fi**$, [], W) =$
if $i \leq \lvert p\rvert$ **then** $(s'{+}{+}lhs(p[i]), w)$ **else** $(st{+}{+}\mathbf{hd}w, \mathbf{tl}w)$ **fi** $\equiv \mathbf{true}$

A.12 Reachable nodes in graph (section 3.4.2)

Instantiations

f	**by**	*reachables*
g	**by**	r
g'	**by**	r'
g''	**by**	r''
X	**by**	(N, G)
x	**by**	(n, g)
y	**by**	m
u	**by**	(n, g)
$r(X, y)$	**by**	m
$\mathrm{Q}(x, y)$	**by**	$reachable(m, n, g)$
$\mathrm{T}(u)$	**by**	$succs(n, g) = []$
$\mathrm{H}(u)$	**by**	$\{n\}$
$\mathrm{D}(u)$	**by**	$\{n\}$
$n(u)$	**by**	$\lvert succs(n, g)\rvert$
$\mathrm{B}(i, u)$	**by**	$\neg visited(g, succs(n,g)[k])$
$\mathrm{K}(i, u)$	**by**	$(succs(n, g)[k], vis(n, g))$
$next(k, u)$	**by**	$k+1$
WF-ORD(**m**, <)	**by**	(**graph**, $g < h \Leftrightarrow \lvert a\text{: }\mathbf{node} \parallel a \in nodes(g) \wedge visited(a, g)\rvert >$ $\lvert a\text{: }\mathbf{node} \parallel a \in nodes(h) \wedge visited(a, h)\rvert$

Proof obligations

(0) $(\lvert\{m\text{: }\mathbf{node} \parallel reachable(m, n, g)\}\rvert < \infty) \equiv \mathbf{true}$

(3) $succs(n, g) = [] \equiv \mathbf{true} \vdash \{n\} \equiv \{m\text{: }\mathbf{node} \parallel reachable(m, n, g)\}$

(4) $succs(n, g) \neq [] \equiv \mathbf{true} \vdash$
$\{m\text{: }\mathbf{node} \parallel reachable(m, n, g)\} \equiv$
$\{n\} \cup \bigcup_{i = 1..\lvert succs(n, g)\rvert} \{m\text{: }\mathbf{node} \parallel \neg visited(g, succs(n, g)[i]) \,\Delta$
$reachable(m, succs(n, g)[i], vis(n, g))\}$

(6) $1 \leq i \leq \lvert succs(n, g)\rvert \,\Delta\, \neg visited(g, succs(n, g)[i]) \equiv \mathbf{true} \vdash \lvert vis(n, g)\rvert > \lvert g\rvert \equiv \mathbf{true}$

A.13 Prime factors of a natural number (section 3.5.2)

Instantiations

f	**by**	*all-primes*
g	**by**	*primes*
g'''	**by**	*prs*
X	**by**	N
x	**by**	n
y	**by**	sp
u	**by**	n
v	**by**	(s, c)
$r(X, y)$	**by**	sp
$\mathrm{Q}(x, y)$	**by**	$prod(sp) = n \wedge ordered(sp)$
$\mathrm{P}(X, u, v)$	**by**	$\forall\ (c'\text{: }\mathbf{nat} \parallel 2 \leq c' < c) \parallel \neg(c' \mid n) \wedge prod(s)*n = N \wedge ordered(s)$
E	**by**	$([], 1)$
$\mathrm{Q}'(u, v, y)$	**by**	$\exists\ s'\text{: }\mathbf{psequ} \parallel sp = s{+}{+}s' \ \wedge prod(s') = n \wedge ordered(sp)$
$\mathrm{T}(u, v)$	**by**	$n = 1$
$\mathrm{H}(u, v)$	**by**	s
$\mathrm{B}(u, v)$	**by**	**true**
$\mathrm{K}(u, v)$	**by**	**if** $c \mid n$ **then** n **div** c **else** n **fi**
$\mathrm{R}(u, v)$	**by**	**if** $c \mid n$ **then** $(s{+}{+}c, c)$ **else** $(s, next(c))$ **fi**
WF-ORD(**m**, <)	**by**	$((\mathbf{psequ}, \mathbf{nat}), (a, b) < (c, d) \ \Leftrightarrow (\lvert a\rvert > \lvert b\rvert) \vee (\lvert a\rvert = \lvert b\rvert \wedge c > d))$

Proof obligations

(0) $(\lvert\{sp\text{: }\mathbf{psequ} \parallel prod(sp) = n \wedge ordered(sp)\}\rvert < \infty) \equiv \mathbf{true}$

(1) $(\forall\ (c'\text{: }\mathbf{nat} \parallel 2 \leq c' < 2) \parallel \neg(c' \mid N) \wedge prod([])*N = N \wedge ordered([])) \equiv \mathbf{true}$

(2) $prod(sp) = n \wedge ordered(sp) \equiv \exists\ s'\text{: }\mathbf{psequ} \parallel sp = []{+}{+}s' \ \wedge prod(s') = n \wedge ordered(sp)$

(3) $(\forall\ (c'\text{: }\mathbf{nat} \parallel 2 \leq c' < c) \parallel \neg(c' \mid n) \wedge prod(s)*n = N \wedge ordered(s))\ \Delta\ n = 1\ \equiv \mathbf{true} \vdash$
$s \equiv \{sp\text{: }\mathbf{psequ} \parallel \exists\ s'\text{: }\mathbf{psequ} \parallel sp = s{+}{+}s' \ \wedge prod(s') = n \wedge ordered(sp)\}$

(4) $(\forall\ (c'\text{: }\mathbf{nat} \parallel 2 \leq c' < c) \parallel \neg(c' \mid n) \wedge prod(s)*n = N \wedge ordered(s))\ \Delta\ n \neq 1\ \equiv \mathbf{true} \vdash$
$\{sp\text{: }\mathbf{psequ} \parallel \exists\ s'\text{: }\mathbf{psequ} \parallel sp = s{+}{+}s' \ \wedge prod(s') = n \wedge ordered(sp)\} \equiv$
$\{sp\text{: }\mathbf{psequ} \parallel \exists\ s'\text{: }\mathbf{psequ} \parallel sp = \mathbf{if}\ c \mid n\ \mathbf{then}\ s{+}{+}c\ \mathbf{else}\ s\ \mathbf{fi}{+}{+}s' \ \wedge$
$prod(s') = \mathbf{if}\ c \mid n\ \mathbf{then}\ n\ \mathbf{div}\ c\ \mathbf{else}\ n\ \mathbf{fi} \wedge ordered(sp)\}$

(5) $(\forall (c'\text{: } \mathbf{nat} \parallel 2 \le c' < c) \parallel \neg(c' \mid n) \wedge prod(s)*n = N \wedge ordered(s))\ \Delta\ n \ne 1 \equiv \mathbf{true} \vdash$
$(\forall (c'\text{: } \mathbf{nat} \parallel 2 \le c' < \mathbf{if}\ c \mid n\ \mathbf{then}\ c\ \mathbf{else}\ next(c)\ \mathbf{fi}) \parallel \neg(c' \mid n) \wedge$
$prod(\mathbf{if}\ c \mid n\ \mathbf{then}\ s{+}{+}c\ \mathbf{else}\ s\ \mathbf{fi}) * \mathbf{if}\ c \mid n\ \mathbf{then}\ n\ \mathbf{div}\ c\ \mathbf{else}\ n\ \mathbf{fi} = N \wedge$
$ordered(\mathbf{if}\ c \mid n\ \mathbf{then}\ s{+}{+}c\ \mathbf{else}\ s\ \mathbf{fi})) \equiv \mathbf{true}$

(6) $\mathbf{if}\ c \mid n\ \mathbf{then}\ (s{+}{+}c, c)\ \mathbf{else}\ (s, next(c))\ \mathbf{fi} < (s, c) \equiv \mathbf{true}$

A.14 All primes up to a given natural number (section 3.5.3)

Instantiations

f	**by**	*all-primes*
g	**by**	*primes*
g'''	**by**	*prs*
X	**by**	N
x	**by**	n
y	**by**	p
u	**by**	n
v	**by**	(q, s)
$r(X, y)$	**by**	p
$Q(x, y)$	**by**	$1 \le p \le n \wedge prime(p)$
$P(X, u, v)$	**by**	$q \le n \wedge s = \{c\text{: } \mathbf{nat} \parallel 1 \le c \le q \wedge prime(c)\}$
E	**by**	$(1, \emptyset)$
$Q'(u, v, y)$	**by**	$p \in s \vee (q \le p \le n \wedge prime(p))$
$T(u, v)$	**by**	$q = n$
$H(u, v)$	**by**	s
$B(u, v)$	**by**	**true**
$K(u, v)$	**by**	n
$R(u, v)$	**by**	$\mathbf{if}\ prime(q)\ \mathbf{then}\ (q+1, s \cup \{q\})\ \mathbf{else}\ (q+1, s)\ \mathbf{fi}$
WF-ORD(**r**, <)	**by**	$(\mathbf{nat}, a < b \Leftrightarrow b > a))$

Proof obligations

(0) $(|\{p\text{: } \mathbf{nat} \parallel 1 \le p \le n \wedge prime(p)\}| < \infty) \equiv \mathbf{true}$

(1) $1 \le N \wedge \emptyset = \{c\text{: } \mathbf{nat} \parallel 1 \le c \le 1 \wedge prime(c)\} \equiv \mathbf{true}$

(2) $1 \le p \le n \wedge prime(p) \equiv p \in \emptyset \vee (1 \le p \le n \wedge prime(p))$

(3) $(q \le n \wedge s = \{c\text{: } \mathbf{nat} \parallel 1 \le c \le q \wedge prime(c)\})\ \Delta\ q = n\ \equiv \mathbf{true} \vdash$
$s \equiv \{p\text{: } \mathbf{nat} \parallel p \in s \vee (q \le p \le n \wedge prime(p))\}$

(4) $(q \leq n \wedge s = \{c\text{: }\mathbf{nat} \parallel 1 \leq c \leq q \wedge prime(c)\})\ \Delta\ q \neq n \ \equiv \mathbf{true} \vdash$
$\{p\text{: }\mathbf{nat} \parallel p \in s \vee (q \leq p \leq n \wedge prime(p))\} \equiv$
$\{p\text{: }\mathbf{nat} \parallel \mathbf{if}\ prime(q)\ \mathbf{then}\ p \in s \cup \{q\}\ \mathbf{else}\ p \in s\ \mathbf{fi} \vee (q{+}1 \leq p \leq n \wedge prime(p))\}$

(5) $(q \leq n \wedge s = \{c\text{: }\mathbf{nat} \parallel 1 \leq c \leq q \wedge prime(c)\})\ \Delta\ q \neq n \ \equiv \mathbf{true} \vdash$
$(q{+}1 \leq n \wedge \mathbf{if}\ prime(q)\ \mathbf{then}\ s \cup \{q\}\ \mathbf{else}\ s\ \mathbf{fi} = \{c\text{: }\mathbf{nat} \parallel 1 \leq c \leq q{+}1 \wedge prime(c)\}) \equiv \mathbf{true}$

(6) $q{+}1 > q \equiv \mathbf{true}$

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluationof Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic
Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars: Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman, Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-11 *Thomas Beuter, Peter Dadam*
Prinzipien der Replikationskontrolle in verteilten Systemen

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants