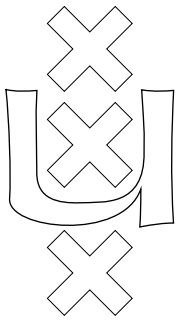
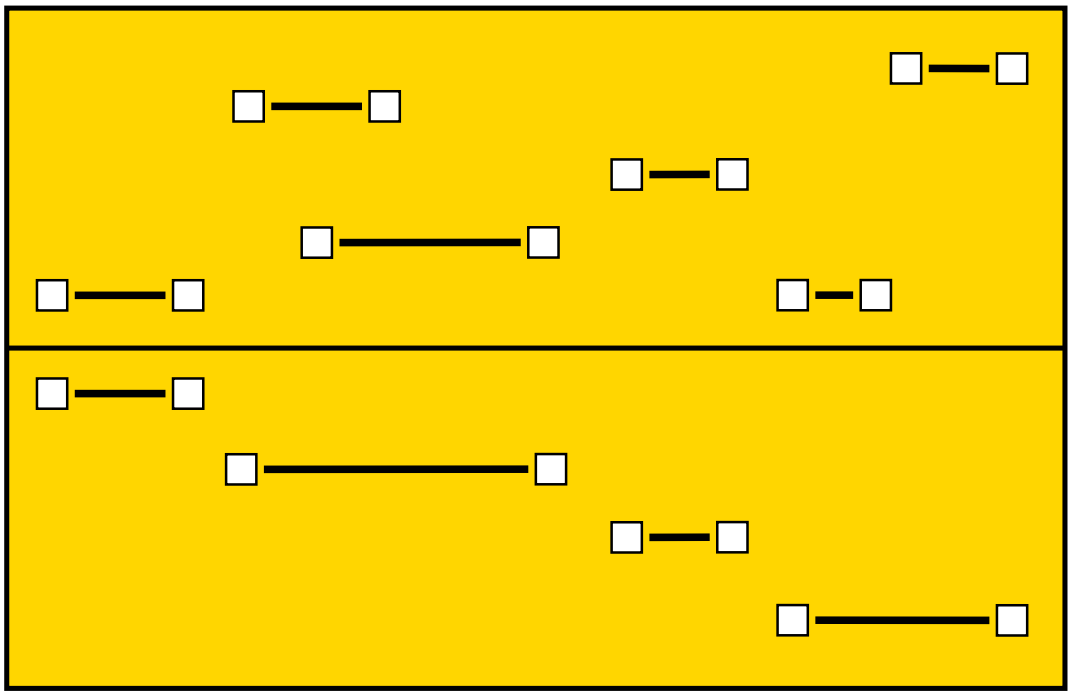


University of Amsterdam

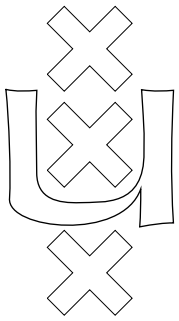
Programming Research Group



Character Classes

Eelco Visser

Report P9708 August 1997



University of Amsterdam

Department of Computer Science

Programming Research Group

Character classes

Eelco Visser

Report P9708 August 1997

E. Visser

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7590
e-mail: visser@wins.uva.nl

This research was supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO). Project 612-317-420: Incremental parser generation and context-sensitive disambiguation: a multi-disciplinary perspective.

Universiteit van Amsterdam, 1997

Character Classes

Eelco Visser

Character classes are used in syntax definition formalisms as compact representations of sets of characters. A character class is a list of characters and ranges of characters. For instance, [A-Z0-9] describes the set containing all uppercase characters and all digits. One set of characters can be represented in many ways with character classes. In this paper an algebraic specification of character classes is presented. We define a normalization of character classes that results in unique, most compact normal forms such that equality of character classes becomes syntactic equality of their normal forms.

1 Introduction

Program or data text is usually written as a list of characters in some character encoding. The syntax of a programming language describes the structure imposed on such lists of characters by the language definition. Character classes are used in syntax definition formalisms as compact representations of sets of characters. A character class is a list of characters and character ranges. For instance, the character class [A-Za-z] denotes the set of all upper- and lowercase letters. Character classes are used in context-free grammars to abbreviate productions. For instance, the lexical production[1]

```
[A-Za-z][A-Za-z0-9]* -> Id
```

defines identifiers to be lists of characters starting with an upper- or lowercase letter and followed by zero or more digits or upper- or lowercase letters. The character classes in this production are implicitly defined by productions

```
"A" -> [A-Za-z]
"B" -> [A-Za-z]
...
"Z" -> [A-Za-z]
"a" -> [A-Za-z]
...
"z" -> [A-Za-z]
```

Operations on character classes are the standard set operations union, difference, intersection and complement with respect to a fixed set of characters. Typical use of such operations is in the lexical production

[1] In the syntax definition formalism SDF (Heering *et al.*, 1989), a context-free production that generates a string α from a non-terminal A is written as $\alpha \rightarrow A$, whereas conventionally this is written as $A \rightarrow \alpha$ or $A ::= \alpha$ in BNF.

```
[\%] [\%] ~[\n]* -> Comment
```

that defines comments as starting with two percent signs followed by zero or more characters from the set of all characters except newline.

Characters can be encoded as numbers. The American Standard Code for Information Interchange is a 7 bit character set standard established by ANSI for representing alphanumeric and control characters. We will use this encoding to uniquely identify characters and character ranges. Using ASCII the character class [A-Za-z] becomes `[\65-\90\97-\122]`. However, this encoding is not a crucial part of the techniques discussed in this paper. Other encodings and larger character sets can also be accommodated.

Each set of characters can be represented by character classes in many ways. For instance, [A-Za-z] is equivalent to [a-zA-Z] and to [A-Hk-zC-Za-l]. In order to establish equality of classes one might test membership. A more efficient way is to transform a class to a unique normal form, i.e., to find a unique representation for each set of characters, by ordering the ranges in a class and combining overlapping or neighboring ranges.

In this paper we give an algebraic specification of characters and character classes and describe how character classes can be normalized to unique, most compact normal forms. This includes a translation of characters to their numeric encoding and simple arithmetic on numeric character codes. We define the equivalence of two character classes and show that any two equivalent character classes have the same normal form.

Requirements on the definition are (1) orthogonal and unrestrictive syntax, i.e., all syntactically correct expressions should have a meaning and (2) efficient rewriting implementation of the transformation.

1.1 Related Work

Character classes were introduced in many contexts as informal abbreviations of a number of productions in a context-free grammar. The oldest explicit description of character classes that we have found is in the manual of the lexical syntax definition formalism LEX (Lesk and Schmidt, 1986)—that stems from the early 70s—character classes are used in regular expressions. Here we start from character classes in the syntax definition formalism SDF (Heering *et al.*, 1989). We have changed the definition to fit in the redesign of SDF described in Visser (1995, 1997a). A predecessor of the specification in this paper appeared in Visser (1994). See Section 9 for a comparison of these definitions.

The specification of character classes is written in the algebraic specification formalism ASF+SDF. An introduction to and further literature on ASF+SDF can be found in Van Deursen *et al.* (1996).

1.2 Overview

In Section 2 we define the syntax of numeric character codes and the syntax of ASCII shortcuts for these codes. In Section 3 we define the normalization of these ASCII shortcuts to their numeric equivalents. In Section 4 we define some simple arithmetic on numeric character codes. In Section 5 we define the syntax of character classes and several operations on character classes. In Section 7 we define the normalization of character classes. In Section 8 sets of

character classes are introduced and are applied to partitioning character classes into mutually disjoint sets.

2 Characters

A character is a constant of the form $\backslash d_1 \ldots d_n$, where the d_i are decimal digits, denoting the $d_1 \ldots d_n$-th member of some finite, linearly ordered universe of characters. Since specifying characters by their index in some encoding scheme is difficult, we provide easier syntax for specification of characters. Alphanumeric characters (letters and digits) can be specified as themselves. Other visible characters in the ASCII set can be specified by escaping them using a backslash, e.g., `\(` for left parenthesis, `\-` for a hyphen and `\ ` (a backslash followed by a space) for space. The characters `\t` and `\n` represent tabs and newlines. Finally, there are two special characters, `\EOF` and `\TOP`. `\EOF` is the character used to indicate represent the end of a file. `\TOP` is used to represent the largest character in the character universe.

module Character-Syntax
imports Layout
exports
  sorts Character NumChar ShortChar
  lexical syntax
    [\][0-9]+ → NumChar
    [a-zA-Z0-9] → ShortChar
    [\]~[\000-\037A-Za-mo-su-z0-9] → ShortChar
  context-free syntax
    NumChar → Character
    ShortChar → Character
    "\TOP" → Character
    "\EOF" → Character
  variables
    "c"[*0-9'*]* → Character

Remark In SDF numeric character codes in character classes are interpreted as octal numbers, whereas in the character classes defined in this paper the decimal encoding is used. Therefore, the definition above defines as short characters all symbols composed of a backslash followed by a character *not* in the range `\0-\37`, i.e, all ASCII characters until and not including the space character.

3 Character Normalization

The short characters introduced in the previous section are mnemonic shortcuts for numeric character codes, according to the following rules. (1) Each printable ASCII character preceded by a backslash (\) denotes its ASCII number, except for `\t` and `\n` that denote tab and newline. (2) The letters A–Z and a–z and the digits 0–9 denote their ASCII numbers. The following equations translate short characters to numeric characters according to these rules and the ASCII encoding scheme.

module Character-Normalization
imports Character-Syntax[2]

equations
Leading zeros are redundant.

[1] $$\text{numchar}(\texttt{"\" "0"}\ c^+) = \text{numchar}(\texttt{"\"}\ c^+)$$

The constant `\TOP` denotes the highest character. In this setting we define top to be `\255` such that all byte represented characters can be handled.

[2] $$\texttt{\TOP} = \texttt{\255}$$
[3] $$\texttt{\EOF} = \texttt{\0}$$

All ASCII symbols preceded by a backslash are translated to their corresponding ASCII code. Digits and letters have the usual encoding except for `\t` and `\n`, which conventionally denote tab and newline.

[4] $$\texttt{\t} = \texttt{\9}$$ [5] $$\texttt{\n} = \texttt{\10}$$

[6]	$\texttt{\␣} = \texttt{\32}$	[7]	$\texttt{\!} = \texttt{\33}$	[8]	$\texttt{\"} = \texttt{\34}$
[9]	$\texttt{\#} = \texttt{\35}$	[10]	$\texttt{\$} = \texttt{\36}$	[11]	$\texttt{\%} = \texttt{\37}$
[12]	$\texttt{\&} = \texttt{\38}$	[13]	$\texttt{\'} = \texttt{\39}$	[14]	$\texttt{\(} = \texttt{\40}$
[15]	$\texttt{\)} = \texttt{\41}$	[16]	$\texttt{*} = \texttt{\42}$	[17]	$\texttt{\+} = \texttt{\43}$
[18]	$\texttt{\,} = \texttt{\44}$	[19]	$\texttt{\-} = \texttt{\45}$	[20]	$\texttt{\.} = \texttt{\46}$
[21]	$\texttt{\/} = \texttt{\47}$				

[22]	$0 = \texttt{\48}$	[23]	$1 = \texttt{\49}$	[24]	$2 = \texttt{\50}$
[25]	$3 = \texttt{\51}$	[26]	$4 = \texttt{\52}$	[27]	$5 = \texttt{\53}$
[28]	$6 = \texttt{\54}$	[29]	$7 = \texttt{\55}$	[30]	$8 = \texttt{\56}$
[31]	$9 = \texttt{\57}$				

[32]	$\texttt{\:} = \texttt{\58}$	[33]	$\texttt{\;} = \texttt{\59}$	[34]	$\texttt{\<} = \texttt{\60}$
[35]	$\texttt{\=} = \texttt{\61}$	[36]	$\texttt{\>} = \texttt{\62}$	[37]	$\texttt{\?} = \texttt{\63}$
[38]	$\texttt{\@} = \texttt{\64}$				

[39]	$\texttt{A} = \texttt{\65}$	[40]	$\texttt{B} = \texttt{\66}$	[41]	$\texttt{C} = \texttt{\67}$
[42]	$\texttt{D} = \texttt{\68}$	[43]	$\texttt{E} = \texttt{\69}$	[44]	$\texttt{F} = \texttt{\70}$
[45]	$\texttt{G} = \texttt{\71}$	[46]	$\texttt{H} = \texttt{\72}$	[47]	$\texttt{I} = \texttt{\73}$
[48]	$\texttt{J} = \texttt{\74}$	[49]	$\texttt{K} = \texttt{\75}$	[50]	$\texttt{L} = \texttt{\76}$
[51]	$\texttt{M} = \texttt{\77}$	[52]	$\texttt{N} = \texttt{\78}$	[53]	$\texttt{O} = \texttt{\79}$
[54]	$\texttt{P} = \texttt{\80}$	[55]	$\texttt{Q} = \texttt{\81}$	[56]	$\texttt{R} = \texttt{\82}$
[57]	$\texttt{S} = \texttt{\83}$	[58]	$\texttt{T} = \texttt{\84}$	[59]	$\texttt{U} = \texttt{\85}$
[60]	$\texttt{V} = \texttt{\86}$	[61]	$\texttt{W} = \texttt{\87}$	[62]	$\texttt{X} = \texttt{\88}$
[63]	$\texttt{Y} = \texttt{\89}$	[64]	$\texttt{Z} = \texttt{\90}$		

[65]	$\texttt{\[} = \texttt{\91}$	[66]	$\texttt{\\} = \texttt{\92}$	[67]	$\texttt{\]} = \texttt{\93}$
[68]	$\texttt{\^} = \texttt{\94}$	[69]	$\texttt{\_} = \texttt{\95}$	[70]	$\texttt{\`} = \texttt{\96}$

[71]	$\texttt{a} = \texttt{\97}$	[72]	$\texttt{b} = \texttt{\98}$	[73]	$\texttt{c} = \texttt{\99}$
[74]	$\texttt{d} = \texttt{\100}$	[75]	$\texttt{e} = \texttt{\101}$	[76]	$\texttt{f} = \texttt{\102}$

[77]	`g = \103`	[78]	`h = \104`	[79]	`i = \105`
[80]	`j = \106`	[81]	`k = \107`	[82]	`l = \108`
[83]	`m = \109`	[84]	`n = \110`	[85]	`o = \111`
[86]	`p = \112`	[87]	`q = \113`	[88]	`r = \114`
[89]	`s = \115`	[90]	`t = \116`	[91]	`u = \117`
[92]	`v = \118`	[93]	`w = \119`	[94]	`x = \120`
[95]	`y = \121`	[96]	`z = \122`		

[97]	`\{ = \123`	[98]	`| = \124`	[99]	`\} = \125`
[100]	`\~ = \126`				

4 Character Arithmetic

We define some simple arithmetic functions on characters, namely, predecessor and successor functions, maximum and minimum of two characters and the inequality relations $<$ and $\leq$. These functions will be needed to compute normal forms of character ranges. The functions are defined by translating character codes to integers and using the appropriate integer operations to perform the computation. See Appendix B.3 for the specification of integers.

module Character-Arithmetic
imports Character-Normalization[3] Integers[B.3]
exports
context-free syntax

int(Character)	→ NatCon
char(Int)	→ Character
pred(Character)	→ Character
succ(Character)	→ Character
Character "≡" Character	→ Bool
Character "≤" Character	→ Bool
Character "<" Character	→ Bool
max(Character, Character)	→ Character
min(Character, Character)	→ Character

equations

[1] $\text{int}(\text{numchar}(\texttt{"\textbackslash"}\ c^+)) = \text{natcon}(c^+)$
[2] $\text{char}(\text{natcon}(c^+)) = \text{numchar}(\texttt{"\textbackslash"}\ c^+)$

[3] $\text{pred}(\mathsf{c}) = \text{char}(\text{int}(\mathsf{c}) - 1)$
[4] $\text{succ}(\mathsf{c}) = \text{char}(\text{int}(\mathsf{c}) + 1)$

Maximum and minimum: using the comparison operators.

[5] $\max(\mathsf{c}_1, \mathsf{c}_2) = \mathsf{c}_1$ **when** $\mathsf{c}_2 < \mathsf{c}_1 = \top$
[6] $\max(\mathsf{c}_1, \mathsf{c}_2) = \mathsf{c}_2$ **otherwise**
[7] $\min(\mathsf{c}_1, \mathsf{c}_2) = \mathsf{c}_1$ **when** $\mathsf{c}_1 < \mathsf{c}_2 = \top$
[8] $\min(\mathsf{c}_1, \mathsf{c}_2) = \mathsf{c}_2$ **otherwise**

Characters can be compared with the relational operators $=$, $<$ and $\leq$. A character c_1 is $\leq$ than a character c_2 if they are equal or if c_1 is less than c_2

[9] $\quad c_1 < c_2 = \bot$ **when** pred(\000) $= c_2$
[10] $\quad c_1 < c_2 = \text{int}(c_1) < \text{int}(c_2)$ **otherwise**
[11] $\quad c \leq c = \top$
[12] $\quad c_1 \leq c_2 = c_1 < c_2$ **when** $c_1 \neq c_2$
[13] $\quad c \equiv c = \top$
[14] $\quad c_1 \equiv c_2 = \bot$ **otherwise**

5 Character Classes

A character class is represented by a list of character ranges. A range is either a single character or a pair of characters connected with a hyphen denoting the characters in the inclusive interval between the two boundaries. A list of ranges is either a single range or the concatenation of two lists. The list of ranges in a class is optional, i.e., can be empty. Operations on character classes are complement $\sim$, difference /, intersection $\wedge$, and union $\vee$.

module Character-Class-Syntax
imports Character-Syntax[2]
exports
sorts CharRange CharRanges OptCharRanges CharClass
context-free syntax

Character	→ CharRange	
Character "–" Character	→ CharRange	
CharRange	→ CharRanges	
CharRanges CharRanges	→ CharRanges	{**right**}
"(" CharRanges ")"	→ CharRanges	{**bracket**}
	→ OptCharRanges	
CharRanges	→ OptCharRanges	
"[" OptCharRanges "]"	→ CharClass	
"∼" CharClass	→ CharClass	
CharClass "/" CharClass	→ CharClass	{**left**}
CharClass "∧" CharClass	→ CharClass	{**left**}
CharClass "∨" CharClass	→ CharClass	{**left**}
"(" CharClass ")"	→ CharClass	{**bracket**}

priorities
"∼"CharClass → CharClass > CharClass "/"CharClass → CharClass
> CharClass "∧"CharClass → CharClass >
CharClass "∨"CharClass → CharClass
variables

"*cr*"[*0-9′*]*	→ CharRange
"*cr*" "*" [*0-9′*]*	→ OptCharRanges
"*cr*" "+" [*0-9′*]*	→ CharRanges
"*cc*"[*0-9′*]*	→ CharClass

6 Membership

The semantics of character classes is defined by means of the predicate $\in$ that determines whether a character is a member of a character class. We can then define that a character class cc corresponds to the set $\{c|c \in cc\}$. The crucial equation is equation [in3] that defines the membership of a character in a range. Furthermore, concatenation of ranges is union (disjunction) and complement is defined with respect to the class $[\backslash 0 - \backslash \mathtt{TOP}]$. The subset relation and equivalence on character classes can then be defined in terms of $\in$. The predicate $\subseteq$ characterizes the inclusion of a class in another class, i.e., a class c_1 is included in a class c_2 if all characters in c_1 are also contained in c_2.

module Character-Class-Membership
imports Character-Class-Syntax[5] Character-Arithmetic[4] Booleans[B.1]
exports
context-free syntax
Character "$\in$" OptCharRanges $\rightarrow$ Bool
Character "$\in$" CharClass $\rightarrow$ Bool
CharClass "$\subseteq$" CharClass $\rightarrow$ Bool
CharClass "$\equiv$" CharClass $\rightarrow$ Bool
equations
List of ranges: range corresponds to interval and concatenation corresponds to union.

[1] $\mathsf{c} \in \mathsf{c} = \top$
[2] $\mathsf{c} \in \mathsf{c}' = \bot$ **when** $\mathsf{c} \neq \mathsf{c}'$
[3] $\mathsf{c} \in \mathsf{c}_1 - \mathsf{c}_2 = \mathsf{c}_1 \leq \mathsf{c} \wedge \mathsf{c} \leq \mathsf{c}_2$
[4] $\mathsf{c} \in \ = \bot$
[5] $\mathsf{c} \in cr_1^+ \ cr_2^+ = \mathsf{c} \in cr_1^+ \vee \mathsf{c} \in cr_2^+$

Character classes: membership of character class is defined in terms of membership of its range list.

[6] $\mathsf{c} \in [cr^*] = \mathsf{c} \in cr^*$

Boolean operations: complement is defined with respect to the range $\backslash 0 - \backslash \mathtt{TOP}$.

[7] $\mathsf{c} \in \sim cc = \mathsf{c} \in [\backslash 000 - \backslash \mathtt{TOP}] \wedge \sim \mathsf{c} \in cc$
[8] $\mathsf{c} \in cc_1 \ / \ cc_2 = \mathsf{c} \in cc_1 \wedge \sim \mathsf{c} \in cc_2$
[9] $\mathsf{c} \in cc_1 \wedge cc_2 = \mathsf{c} \in cc_1 \wedge \mathsf{c} \in cc_2$
[10] $\mathsf{c} \in cc_1 \vee cc_2 = \mathsf{c} \in cc_1 \vee \mathsf{c} \in cc_2$

Inclusion: all characters in subset are elements of the superset.

[11] $[] \subseteq cc = \top$
[12] $[\mathsf{c}] \subseteq cc = \mathsf{c} \in cc$
[13] $[\mathsf{c} - \mathsf{c}] \subseteq cc = \mathsf{c} \in cc$
[14] $[\mathsf{c}_1 - \mathsf{c}_2] \subseteq cc = \mathsf{c}_1 \in cc \wedge [\mathrm{succ}(\mathsf{c}_1) - \mathsf{c}_2] \subseteq cc$ **when** $\mathsf{c}_1 < \mathsf{c}_2 = \top$
[15] $[cr_1^+ \ cr_2^+] \subseteq cc = [cr_1^+] \subseteq cc \wedge [cr_2^+] \subseteq cc$

Equivalence: two character classes are equivalent if both are subsets of each other.

[16] $cc_1 \equiv cc_2 = cc_1 \subseteq cc_2 \wedge cc_2 \subseteq cc_1$

Later we will also formulate equivalence as $\forall c : c \in cc_1 \iff c \in cc_2$, which is equivalent to [eqv].

7 Character Class Normalization

In this section we define equations for range list concatenation that reduce range lists such that two classes are equivalent if and only if they have the same normal form, i.e., $\forall cc_1, cc_2 : cc_1 \equiv cc_2 \iff \exists cc_3 : cc_1 \twoheadrightarrow cc_3 \twoheadleftarrow cc_2$.

An obvious normal form would be to expand all ranges to lists of characters, and sort the resulting set of characters. However, with large character sets this is an expensive representation. The normal form of a character class defined below consists of a list of characters and ranges such that the ranges are mutually non-overlapping and such that they are sorted in increasing order. For example, the normal form of `[A-Z0-9\%z-a]` is `[\37\48-\57\65-\90]`. This is the most compact representation of a set of characters. In the worst case, when every other character is included in a class, it has size $n/2$ with n the number of characters. But in most cases this will be much smaller. The expansion method always produces a class the size of the number of characters in the class.

module Character-Class-Normalization

7.1 Preliminaries

imports Character-Class-Syntax[5] Character-Arithmetic[4]
Character-Class-Membership[6]
hiddens
context-free syntax

"$\emptyset$"	$\rightarrow$ CharRange
if(Bool, CharClass, CharClass)	$\rightarrow$ CharClass
head(CharRanges)	$\rightarrow$ CharRange
tail(CharRanges)	$\rightarrow$ CharRanges
CharRange "$\prec$" CharRange	$\rightarrow$ Bool
CharRange "$\not\prec$" CharRange	$\rightarrow$ Bool
CharRange "$\not\not\prec$" CharRange	$\rightarrow$ Bool
sorted(OptCharRanges)	$\rightarrow$ Bool
sorted(CharClass)	$\rightarrow$ Bool

equations
A range from a character to itself denotes just the set containing that character.

[1] $$\mathsf{c} - \mathsf{c} = \mathsf{c}$$

The constant range $\emptyset$ denotes an empty range, i.e., a set containing no characters. It can thus be removed from a class. The constant is used to normalize degenerate ranges like $\backslash 10 - \backslash 9$ that are empty.

[2] $$\mathsf{c}_1 - \mathsf{c}_2 = \emptyset \quad \textbf{when } \mathsf{c}_1 \leq \mathsf{c}_2 = \bot$$

From now on we will assume that all ranges $c_1 - c_2$ in left-hand sides of equations are proper, i.e., $c_1 < c_2$.

Conditional: choose between two character classes depending on condition.

[3] $$\text{if}(\top,\, cc_1,\, cc_2) = cc_1$$

[4] $\text{if}(\bot, cc_1, cc_2) = cc_2$

The head and tail of a list of character ranges.

[5] $\text{head}(cr) = cr$ [6] $\text{tail}(cr) = \emptyset$
[7] $\text{head}(cr\ cr^+) = cr$ [8] $\text{tail}(cr\ cr^+) = cr^+$

A range cr_1 is *left-smaller* than a range cr_2 if cr_1's left edge is smaller than cr_2's left edge. We have, for instance, that $\texttt{\textbackslash 10} - \texttt{\textbackslash 35} \prec \texttt{\textbackslash 16} - \texttt{\textbackslash 92}$.

[9] $\mathsf{c}_1 \prec \mathsf{c}_3 = \mathsf{c}_1 < \mathsf{c}_3$
[10] $\mathsf{c}_1 \prec \mathsf{c}_3 - \mathsf{c}_4 = \mathsf{c}_1 < \mathsf{c}_3$
[11] $\mathsf{c}_1 - \mathsf{c}_2 \prec \mathsf{c}_3 = \mathsf{c}_1 < \mathsf{c}_3$
[12] $\mathsf{c}_1 - \mathsf{c}_2 \prec \mathsf{c}_3 - \mathsf{c}_4 = \mathsf{c}_1 < \mathsf{c}_3$

A range cr_1 is *smaller* than a range cr_2 if cr_1's right edge is smaller than cr_2's left edge. For example, $\texttt{\textbackslash 10} - \texttt{\textbackslash 35} \nprec \texttt{\textbackslash 36} - \texttt{\textbackslash 92}$.

[13] $\mathsf{c}_1 \nprec \mathsf{c}_3 = \mathsf{c}_1 < \mathsf{c}_3$
[14] $\mathsf{c}_1 \nprec \mathsf{c}_3 - \mathsf{c}_4 = \mathsf{c}_1 < \mathsf{c}_3$
[15] $\mathsf{c}_1 - \mathsf{c}_2 \nprec \mathsf{c}_3 = \mathsf{c}_2 < \mathsf{c}_3$
[16] $\mathsf{c}_1 - \mathsf{c}_2 \nprec \mathsf{c}_3 - \mathsf{c}_4 = \mathsf{c}_2 < \mathsf{c}_3$

A range cr_1 is *strictly smaller* than a range cr_2 if cr_1's right edge is at least one smaller than cr_2's left edge. For example, $\texttt{\textbackslash 10} - \texttt{\textbackslash 34} \nprec\nprec \texttt{\textbackslash 36} - \texttt{\textbackslash 92}$, but not $\texttt{\textbackslash 10} - \texttt{\textbackslash 35} \nprec\nprec \texttt{\textbackslash 36} - \texttt{\textbackslash 92}$. This entails that if $cr_1 \nprec\nprec cr_2$, the ranges cr_1 and cr_2 do not overlap.

[17] $\mathsf{c}_1 \nprec\nprec \mathsf{c}_3 = \text{succ}(\mathsf{c}_1) < \mathsf{c}_3$
[18] $\mathsf{c}_1 \nprec\nprec \mathsf{c}_3 - \mathsf{c}_4 = \text{succ}(\mathsf{c}_1) < \mathsf{c}_3$
[19] $\mathsf{c}_1 - \mathsf{c}_2 \nprec\nprec \mathsf{c}_3 = \text{succ}(\mathsf{c}_2) < \mathsf{c}_3$
[20] $\mathsf{c}_1 - \mathsf{c}_2 \nprec\nprec \mathsf{c}_3 - \mathsf{c}_4 = \text{succ}(\mathsf{c}_2) < \mathsf{c}_3$

A list of ranges is *sorted* if each range is strictly smaller than its successor. A character class is sorted if its list of ranges is sorted.

[21] $\text{sorted}() = \top$
[22] $\text{sorted}(cr) = \top$
[23] $\text{sorted}(cr_1\ cr_2) = cr_1 \nprec\nprec cr_2$
[24] $\text{sorted}(cr_1\ cr_2\ cr^+) = cr_1 \nprec\nprec cr_2 \wedge \text{sorted}(cr_2\ cr^+)$
[25] $\text{sorted}([cr^*]) = \text{sorted}(cr^*)$

7.2 Sorting and Merging Range Lists

We define normalization of the concatenation of character ranges such that its normal forms are always sorted.

hiddens
  sorts Result
  context-free syntax
    "<" CharRange ">" → Result
    CharRange "▷" CharRange → Result

equations

First we define the operator $\rhd$ that merges to overlapping ranges, provided that the first is smaller than the second. The following five cases have to be considered.

[26] $$\mathsf{c} \rhd \mathsf{c} \;=\; \langle \mathsf{c} \rangle$$

[27] $$\frac{\mathrm{succ}(\mathsf{c}_1) = \mathsf{c}_2}{\mathsf{c}_1 \rhd \mathsf{c}_2 \;=\; \langle \mathsf{c}_1 - \mathsf{c}_2 \rangle}$$

[28] $$\frac{\mathsf{c}_1 \leq \mathsf{c}_3 = \top,\ \mathsf{c}_3 \leq \mathrm{succ}(\mathsf{c}_2) = \top}{\mathsf{c}_1 - \mathsf{c}_2 \rhd \mathsf{c}_3 \;=\; \langle \mathsf{c}_1 - \max(\mathsf{c}_2, \mathsf{c}_3) \rangle}$$

[29] $$\frac{\mathsf{c}_1 \equiv \mathsf{c}_2 \vee \mathrm{succ}(\mathsf{c}_1) \equiv \mathsf{c}_2 = \top}{\mathsf{c}_1 \rhd \mathsf{c}_2 - \mathsf{c}_3 \;=\; \langle \mathsf{c}_1 - \mathsf{c}_3 \rangle}$$

[30] $$\frac{\mathsf{c}_1 \leq \mathsf{c}_3 = \top,\ \mathsf{c}_3 \leq \mathrm{succ}(\mathsf{c}_2) = \top}{\mathsf{c}_1 - \mathsf{c}_2 \rhd \mathsf{c}_3 - \mathsf{c}_4 \;=\; \langle \mathsf{c}_1 - \max(\mathsf{c}_2, \mathsf{c}_4) \rangle}$$

Using this operation we can define the normalization of a concatenation of ranges. In the first place, the empty range is a unit for concatenation and a class with only an empty range is equal to the empty class.

[31] $$[\emptyset] = []$$
[32] $$\emptyset\ cr^+ = cr^+$$
[33] $$cr^+\ \emptyset = cr^+$$

The steps involved in normalizing a list of character ranges are

- make the list right associative
- order the ranges with respect to $\prec$
- merge overlapping ranges.

We could achieve this by means of the following equations.

$$(cr_1^+\ cr_2^+)\ cr_3^+ = cr_1^+\ (cr_2^+\ cr_3^+)$$

$$\frac{cr_2 \prec cr_1 = \top}{cr_1\ cr_2 = cr_2\ cr_1} \qquad \frac{cr_2 \prec cr_1 = \top}{cr_1\ (cr_2\ cr^+) = cr_2\ (cr_1\ cr^+)}$$

$$\frac{cr_1 \rhd cr_2 = \langle cr_3 \rangle}{cr_1\ cr_2 = cr_3} \qquad \frac{cr_1 \rhd cr_2 = \langle cr_3 \rangle}{cr_1\ (cr_2\ cr^+) = cr_3\ cr^+}$$

However, if we assume that the lists that are being concatenated are already in normal form, we can achieve a more efficient definition of concatenation. This is formalized in the next three equations. The head range of the first list is only made the head range of the new list if it is strictly smaller than the head of the second list. (By induction it is already strictly smaller than the head of the rest of its own list.) The second equation commutes the arguments of a concatenation if the second is smaller than the first. The third equation merges

the heads of two lists into a single range and then concatenates the result with the tail of the first list.

[34] $$\frac{cr_1 \lll \mathrm{head}(cr_3^+) = \top}{(cr_1\ cr_2^+)\ cr_3^+ \ = \ cr_1\ cr_2^+\ cr_3^+}$$

[35] $$\frac{\mathrm{head}(cr_2^+) \prec \mathrm{head}(cr_1^+) = \top}{cr_1^+\ cr_2^+ \ = \ cr_2^+\ cr_1^+}$$

[36] $$\frac{\mathrm{head}(cr_1^+) \rhd \mathrm{head}(cr_2^+) = \langle cr_3 \rangle}{cr_1^+\ cr_2^+ \ = \ (cr_3\ \mathrm{tail}(cr_1^+))\ \mathrm{tail}(cr_2^+)}$$

Union of character classes is now defined by concatenating the range lists of the two classes.

[37] $$[] \vee cc = cc$$
[38] $$cc \vee [] = cc$$
[39] $$[cr_1^+] \vee [cr_2^+] = [cr_1^+\ cr_2^+]$$

7.3 Difference

The difference cc_1/cc_2 of two classes cc_1 and cc_2 is the class cc_3 that contains all characters that are in cc_1, but not in cc_2. We introduce the following auxiliary function.

hiddens
context-free syntax
CharRange "/" CharRange → CharClass
equations

First we define the difference of single character ranges.

[40] $$\mathsf{c}_1 \ / \ \mathsf{c}_2 \ = \ \mathrm{if}(\mathsf{c}_1 \equiv \mathsf{c}_2, [], [\mathsf{c}_1])$$

[41] $$\mathsf{c}_1 \ / \ \mathsf{c}_3 - \mathsf{c}_4 \ = \ \mathrm{if}(\mathsf{c}_3 \leq \mathsf{c}_1 \wedge \mathsf{c}_1 \leq \mathsf{c}_4, [], [\mathsf{c}_1])$$

[42] $$\mathsf{c}_1 - \mathsf{c}_2 \ / \ \mathsf{c}_3 \ = \ \mathrm{if}(\mathsf{c}_1 \leq \mathsf{c}_3 \wedge \mathsf{c}_3 \leq \mathsf{c}_2, [\mathsf{c}_1 - \mathrm{pred}(\mathsf{c}_3)\ \mathrm{succ}(\mathsf{c}_3) - \mathsf{c}_2], [\mathsf{c}_1 - \mathsf{c}_2])$$

[43] $$\frac{\mathrm{succ}(\mathsf{c}_4) = \mathsf{c}_4'}{\mathsf{c}_1 - \mathsf{c}_2 \ / \ \mathsf{c}_3 - \mathsf{c}_4 \ = \ \mathrm{if}(\mathsf{c}_3 \leq \mathsf{c}_2 \wedge \mathsf{c}_1 < \mathsf{c}_4', [\mathsf{c}_1 - \mathrm{pred}(\mathsf{c}_3)\ \mathsf{c}_4' - \mathsf{c}_2], [\mathsf{c}_1 - \mathsf{c}_2])}$$

If either character class is empty, the difference is immediate.

[44] $$[] \ / \ cc = []$$
[45] $$cc \ / \ [] = cc$$

If there is no overlap between the first two ranges in the list, then the smallest can be skipped.

[46] $$\frac{\mathrm{head}(cr_2^+) \lll \mathrm{head}(cr_1^+) = \top}{[cr_1^+] \ / \ [cr_2^+] \ = \ [cr_1^+] \ / \ [\mathrm{tail}(cr_2^+)]}$$

[47]
$$\frac{\mathrm{head}(cr_1^+) \not\prec \mathrm{head}(cr_2^+) = \top}{[cr_1^+] \mathbin{/} [cr_2^+] \;=\; [\mathrm{head}(cr_1^+)] \vee [\mathrm{tail}(cr_1^+)] \mathbin{/} [cr_2^+]}$$

If the conditions of the previous two equations fail, then the first two ranges have an overlap. In this case the differences between these ranges are taken.

[48]
$$\frac{\mathrm{head}(cr_1^+) = cr_1,\ \mathrm{head}(cr_2^+) = cr_2}{\begin{array}{rl}[cr_1^+] \mathbin{/} [cr_2^+] \;=\; & (cr_1 \mathbin{/} cr_2 \vee [\mathrm{tail}(cr_1^+)]) \\ & \mathbin{/} (cr_2 \mathbin{/} cr_1 \vee [\mathrm{tail}(cr_2^+)])\end{array}} \quad \textbf{otherwise}$$

7.4 Other Operations

Now we can define the other operations on character classes. The complement with respect to $[\backslash 0 - \backslash \mathtt{TOP}]$. The complement of classes of more than one range is defined in terms of difference.

[49] $$\sim [] = [\backslash 0 - \backslash \mathtt{TOP}]$$
[50] $$\sim [cr^+] = [\backslash 0 - \backslash \mathtt{TOP}] \mathbin{/} [cr^+]$$

Intersection is defined in terms of difference.

[51] $$cc_1 \wedge cc_2 = cc_1 \mathbin{/} (cc_1 \mathbin{/} cc_2)$$

Equality boils down to syntactic equality. See Appendix A for a justification.

[52] $$cc \equiv cc = \top$$
[53] $$cc_1 \equiv cc_2 = \bot \quad \textbf{when}\ \ cc_1 \neq cc_2$$

A character is a member of a character class if it is contained in one of its ranges.

[54] $$\mathsf{c} \in \mathsf{c} = \top$$
[55] $$\mathsf{c} \in \mathsf{c}' = \bot \quad \textbf{when}\ \ \mathsf{c} \neq \mathsf{c}'$$
[56] $$\mathsf{c} \in \mathsf{c}_1 - \mathsf{c}_2 = \mathsf{c}_1 \leq \mathsf{c} \wedge \mathsf{c} \leq \mathsf{c}_2$$
[57] $$\mathsf{c} \in\ = \bot$$
[58] $$\mathsf{c} \in cr\ cr^+ = \bot \quad \textbf{when}\ \ \mathsf{c} \prec cr = \top$$
[59] $$\mathsf{c} \in cr\ cr^+ = \top \quad \textbf{when}\ \ \mathsf{c} \in cr = \top$$
[60] $$\mathsf{c} \in cr\ cr^+ = \mathsf{c} \in cr^+$$

[61] $$\mathsf{c} \in [cr^*] = \mathsf{c} \in cr^*$$

A character class cc_1 is a subset of a character class cc_2 if the union of the two classes equals cc_2.

[62] $$cc \subseteq cc = \top$$
[63] $$[\mathsf{c}] \subseteq cc = \mathsf{c} \in cc$$
[64] $$cc_1 \subseteq cc_2 = cc_1 \vee cc_2 \equiv cc_2 \quad \textbf{otherwise}$$

In Appendix A we will prove the following lemmas and the correctness theorem they entail. The first main lemma states that the equations for sorting range lists are sound with respect to equivalence.

Lemma 7.1 (Soundness) $[cr_1^*] =_E [cr_2^*] \Rightarrow [cr_1^*] \equiv [cr_2^*]$

The completeness lemma states that every range list reduces to a sorted range list. Two other lemmas state that sorted range lists are in normal form and that no two different sorted range lists are equivalent.

Lemma 7.2 (Completeness) $\forall cr_1^* \exists cr_2^* : \text{sorted}(cr_2^*) \wedge cr_1^* \twoheadrightarrow cr_2^*$

Finally, the correctness theorem states that any two equivalent character classes have the same sorted normal form.

Theorem 7.3 (Correctness) $\forall cc_1, cc_2 : cc_1 \equiv cc_2 \iff \exists cc_3 : cc_1 \twoheadrightarrow cc_3 \twoheadleftarrow cc_2 \wedge \text{sorted}(cc_3)$

This concludes the specification of character classes. In the next section an application of character classes is defined.

8 Sets of Character Classes

As an application we define the data type of sets (or rather lists) of character classes. The difference and intersection operators work on the product of sets of character classes, i.e., they apply the corresponding character class operation to each pair in the cartesion product of the argument sets. These operations are used to define the partioning of sets of character classes. The result is a set of character clases that are pairwise disjunct, as is illustrated in Figure 1. This partioning operator can be applied to determine the smallest partioning of a set of transitions with characters that may overlap. For example, the partitioning of

```
{} : [\t\n\ ] : [A-Za-z] : [t]
```

is

```
{[\116] [\009-\010\032] [\065-\090 \097-\115 \117-\122]}
```

This is used in the parser generator for SDF2 (Visser, 1997b) to compute maximal common lookahead between productions.

module Character-Class-Sets
imports Character-Class-Normalization[7]
exports
  sorts CharClassSet
  context-free syntax
    "{" CharClass* "}" → CharClassSet
    CharClassSet "⧺" CharClassSet → CharClassSet {**assoc**}
    CharClassSet "/" CharClassSet → CharClassSet {**left**}
    CharClassSet "∧" CharClassSet → CharClassSet {**left**}
    CharClassSet ":" CharClass → CharClassSet
    "(" CharClassSet ")" → CharClassSet {**bracket**}
  priorities
    CharClassSet "∧"CharClassSet → CharClassSet >
    CharClassSet "/"CharClassSet → CharClassSet >
    CharClassSet "⧺"CharClassSet → CharClassSet

variables

"cc" "$*$" [*0-9′*]$*$ $\rightarrow$ CharClass$*$
"cc" "$+$" [*0-9′*]$*$ $\rightarrow$ CharClass$+$
"ccs" [*0-9′*]$*$ $\rightarrow$ CharClassSet

equations

Remove empty classes

[1] $$\{cc_1^*\ []\ cc_2^*\} = \{cc_1^*\ cc_2^*\}$$

Concatenation

[2] $$\{cc_1^*\} +\!\!+ \{cc_2^*\} = \{cc_1^*\ cc_2^*\}$$

The difference ccs_1/ccs_2 takes the difference of each class in ccs_1 with all classes in ccs_2.

[3] $$ccs \ / \ \{\} = ccs$$
[4] $$\{\} \ / \ ccs = \{\}$$
[5] $$\{cc\} \ / \ \{cc'\} = \{cc \ / \ cc'\}$$
[6] $$ccs \ / \ \{cc_1^+\ cc_2^+\} = ccs \ / \ \{cc_1^+\} \ / \ \{cc_2^+\}$$
[7] $$\{cc_1^+\ cc_2^+\} \ / \ ccs = \{cc_1^+\} \ / \ ccs +\!\!+ \{cc_2^+\} \ / \ ccs$$

The pairwise intersection $ccs_1 \wedge ccs_2$ is the set of classes ccs_3 that contains the intersection of each pair of classes in ccs_1 and ccs_2.

[8] $$ccs \wedge \{\} = \{\}$$
[9] $$\{\} \wedge ccs = \{\}$$
[10] $$\{cc\} \wedge \{cc'\} = \{cc \wedge cc'\}$$
[11] $$ccs \wedge \{cc_1^+\ cc_2^+\} = ccs \wedge \{cc_1^+\} +\!\!+ ccs \wedge \{cc_2^+\}$$
[12] $$\{cc_1^+\ cc_2^+\} \wedge ccs = \{cc_1^+\} \wedge ccs +\!\!+ \{cc_2^+\} \wedge ccs$$

The partitioning of two sets.

[13] $$ccs : cc = ccs \wedge \{cc\} +\!\!+ ccs \ / \ \{cc\} +\!\!+ \{cc\} \ / \ ccs$$

9 Related Work

In this section we briefly compare our definition with the character classes in LEX and SDF and discuss some previous attempts to specify character classes in ASF+SDF.

Character Classes in LEX LEX is the lexical syntax definition formalism of YACC (Lesk and Schmidt, 1986). Basic character classes look like the ones defined in this paper: a list of characters and character ranges between square brackets. The use of the backslash as escape character is also the same. The differences are: Numeric characters can be specified in the same way but are interpreted as octal numbers. The character ^ as first in the list indicates the complement of the class. The list of ranges is lexical, meaning that spaces are significant and denote the space character. No other operations on character classes than complement are provided.

The character classes in LEX are summarized by the following table (due to Aho *et al.* (1986)):

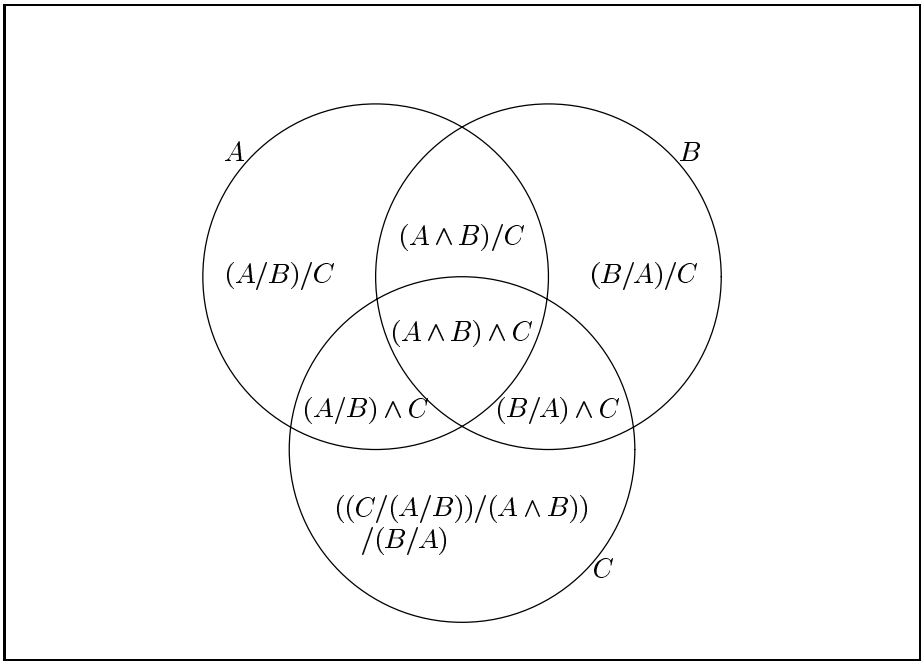


Figure 1: The partitioning of three sets A, B and C. This can be expressed by $\{\} : A : B : C$.

Expression	Matches	Example
c	any non-operator character c	`a`
$\backslash c$	character c literally	`\*`
$\backslash ddd$	character with ASCII code ddd (octal)	`\040`
`.`	any character but newline	`a.*b`
`^`	beginning of line	`^abc`
`$`	end of line	`abc$`
c_1`-`c_2	any character in range $c_1 - c_2$	`A-Z`
`[`s`]`	any character in s	`[abc]`
`[^s]`	any character not in s	`[^abc]`

Character Classes in SDF In SDF character classes are defined lexically as a list of characters between square brackets. This is interpreted after parsing as a list of characters and character ranges. Therefore, the space character is significant. Only complement is provided as operation on character classes. Numeric characters use octal encoding.

10 Concluding Remarks

We have defined an algebraic specification of character classes that defines unique normal forms for character classes. The specification of the redesigned syntax definition formalism SDF (Visser, 1995, 1997a) makes use of the specification of characer classes in this paper. The partitioning of a list of character class is used in the specfication of a parser generator for context-free grammars with character classes (Visser, 1997b).

References

Aho, A., Sethi, R., and Ullman, J. (1986). *Compilers: Principles, techniques, and tools*. Addison Wesley, Reading, Massachusetts.

Van Deursen, A., Heering, J., and Klint, P., editors (1996). *Language Prototyping. An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific, Singapore.

Heering, J., Hendriks, P. R. H., Klint, P., and Rekers, J. (1989). The syntax definition formalism SDF – reference manual. *SIGPLAN Notices*, **24**(11), 43–75.

Lesk, M. E. and Schmidt, E. (1986). *LEX — A lexical analyzer generator*. Bell Laboratories. UNIX Programmer's Supplementary Documents, Volume 1 (PS1).

Visser, E. (1994). Writing course notes with ASF+SDF to LaTeX. In T. B. Dinesh and S. M. Üsküdarlı, editors, *Using the ASF+SDF environment for teaching computer science*, chapter 6. Collection submitted to NSF workshop on teaching formal methods.

Visser, E. (1995). A family of syntax definition formalisms. In M. G. J. van den Brand *et al.*, editors, *ASF+SDF'95. A Workshop on Generating Tools from Algebraic Specifications*, pages 89–126. Technical Report P9504, Programming Research Group, University of Amsterdam.

Visser, E. (1997a). A family of syntax definition formalisms. Technical Report P9706, Programming Research Group, University of Amsterdam.

Visser, E. (1997b). Scannerless generalized-LR parsing. Technical Report P9707, Programming Research Group, University of Amsterdam.

A Correctness

Now we will show that any character class has a unique, most compact normal form that has the same meaning as the original class, i.e., denotes the same set of characters. We use the following notation:

- $=$ indicates literal, syntactic equality
- $\equiv$ indicates equivalence of character classes according to the definition in Section 6.
- $=_E$ indicates equality derivable by equations above
- $\twoheadrightarrow$ indicates reducible by equations above as rewrite rules oriented from left to right
- $\mathrm{nf}(x)$ indicates that x is in normal form with respect to the rewrite rules

First we check the soundness with respect to equivalence of the equations that sort a range list. As an auxiliary result we need the following lemma about the merging of ranges.

Lemma 1.1 $cr_1 \rhd cr_2 =_E \langle cr_3 \rangle \Rightarrow \forall c : (c \in cr_1 \vee c \in cr_2 \iff c \in cr_3)$

Proof. We have to check for each of the equations [26] until [30] that the condition holds. For [26], [27] and [29] this is trivial.

For equation [28] we have the following argument

$$
\begin{array}{ll}
 & c \in c_1 - c_2 \vee c \in c_3 \\
\iff & \text{by definition of range membership} \\
 & (c_1 \leq c \wedge c \leq c_2) \vee c = c_3 \\
\iff & \text{by conditions of [mrg3]} \\
 & c_1 \leq c \wedge c \leq \max(c_2, c_3)
\end{array}
$$

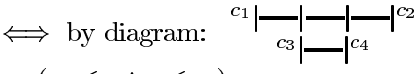
For equation [30] we distinguish two cases. In case (1) range $c_3 - c_4$ partially overlaps and partially extends to the right of the range $c_1 - c_2$, i.e., we have the situation $^{c_1}|\!\!-\!\!-\!\!-\!\!-\!\!|^{c_2}$ $_{c_3}|\!\!-\!\!-\!\!-\!\!-\!\!|_{c_4}$. In case (2) range $c_3 - c_4$ is included in range $c_1 - c_2$, i.e., we have $^{c_1}|\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!|^{c_2}$ $_{c_3}|\!\!-\!\!-\!\!|_{c_4}$. We then argue as follows

$$
\begin{array}{ll}
 & (c \in c_1 - c_2) \vee (c \in c_3 - c_4) \\
\iff & \text{by definition of range membership} \\
 & (c_1 \leq c \wedge c \leq c_2) \vee (c_3 \leq c \wedge c \leq c_4) \\
 & \text{case (1) assume } c_2 < c_4\text{:} \\
\iff & \text{by condition } c_1 \leq c_3 \leq c_2 \\
 & (c_1 \leq c \wedge c \leq c_2) \vee (c_3 \leq c \wedge c \leq c_2) \vee (c_2 \leq c \wedge c \leq c_4) \\
\iff & \text{by diagram: } {}^{c_1}|\!\!-\!\!-\!\!-\!\!-\!\!|^{c_2}\ {}_{c_3}|\!\!-\!\!-\!\!|\!\!-\!\!-\!\!-\!\!-\!\!|_{c_4} \\
 & (c_1 \leq c \wedge c \leq c_2) \vee (c_2 \leq c \wedge c \leq c_4) \\
\iff & \\
 & (c_1 \leq c \wedge c \leq c_4) \\
\iff & \text{by assumption } \max(c_2, c_4) = c_4
\end{array}
$$

$(c_1 \leq c \wedge c \leq \max(c_2, c_4))$

case (2) assume $c_4 \leq c_2$:

$\Longleftrightarrow$ by diagram:



$(c_1 \leq c \wedge c \leq c_2)$

$\Longleftrightarrow$ by assumption $\max(c_2, c_4) = c_2$

$(c_1 \leq c \wedge c \leq \max(c_2, c_4))$

$\Longleftrightarrow$ by definition of range membership for case (1) and (2)

$c \in c_1 - \max(c_2, c_4)$

□

Now we can verify that the equations on range-list concatenation preserve membership.

Lemma 1.2 (Soundness) $cr_1^* =_E cr_2^* \Rightarrow \forall c : (c \in cr_1^* \iff c \in cr_2^*)$

Proof. We have to check that the equations over range concatenation preserve membership of character ranges. For equations [32] and [33] this is clear since $\emptyset$ denotes the empty range. Equations [34] and [35] only permute the order of the ranges and therefore leave the membership intact. For equation [36] we get soundness via the previous lemma. □

From this lemma follows that the equations above preserve character class equivalence.

Corollary 1.3 (Soundness) $[cr_1^*] =_E [cr_2^*] \Rightarrow [cr_1^*] \equiv [cr_2^*]$

Next we show that every character class reduces to a sorted character class that is in normal form. First we show some properties of sortedness. If a character class is sorted it can not be rewritten, i.e., is in normal form.

Lemma 1.4 (Normal Form) $\text{sorted}(cr^*) \Rightarrow \text{nf}(cr^*)$

Proof. If cr^* is sorted it must be right-associative and not contain $\emptyset$s. Furthermore, equation [35] does not apply because $cr_1 \lessdot\!\lessdot cr_2 \Rightarrow \neg(cr_1 \prec cr_2)$. Also equation [36] does not apply because $cr_1 \lessdot\!\lessdot cr_2$ implies that cr_1 and cr_2 have no overlap. □

Lemma 1.5 *The smallest element of a character class is the first element in the sorted range list.*

The next lemma shows that there is exactly one unique sorted character class per equivalence class.

Lemma 1.6 (Unique) $\text{sorted}(cr_1^*) \wedge \text{sorted}(cr_2^*) \wedge [cr_1^*] \equiv [cr_2^*] \Rightarrow cr_1^* = cr_2^*$

Proof. By induction on the lengths n and m of the lists $X = cr_1^*$ and $Y = cr_1^*$.

$(n = 0)$ then m must also be 0.

$(n + 1)$: by induction on m:

$(m = 0)$ Contradiction with equivalence.

$(m+1)$ Consider the first range in X and Y. Their first element must be the same (previous lemma). If their last element is also the same, then, by induction hypothesis, we have that the rest of the ranges are also the same. On the other hand, if the right edge of $X_1 = c_1 - c_2$ is not equal to right edge of Y_1 then take $c = \text{succ}(c_2)$. Then $c \notin X$ and by equivalence $c \notin Y$. □

Lemma 1.7 (Merge) $\neg(cr_2 \prec cr_1) \wedge \neg(cr_1 \nprec cr_2) \iff cr_1 \rhd cr_2 \rightarrow \langle cr_3 \rangle$

Proof. For each combination of ranges we have to check this. We examine the case of two ranges, i.e., $cr_1 = c_1 - c_2$ and $cr_2 = c_3 - c_4$. The premisses $\neg(cr_2 \prec cr_1) \wedge \neg(cr_1 \nprec cr_2)$ then entail $\neg(c_3 < c_1) \Rightarrow c_1 \leq c_3$ and $\neg(\text{succ}(c_2) < c_3) \Rightarrow c_3 \leq \text{succ}(c_2)$, which fullfills the conditions of equation [30] such that $cr_1 \rhd cr_2 \rightarrow \langle c_1 - \max(c_2, c_4) \rangle$. □

Any character class reduces to a sorted character class. Since the equations, and thus the rewrite rules, preserve equivalence, we have that each character class is equivalent to a sorted character class.

Lemma 1.8 (Completeness) $\forall cr_1^* \exists cr_2^* : \text{sorted}(cr_2^*) \wedge cr_1^* \twoheadrightarrow cr_2^*$

Proof. (Sketch) If cr_1^* is the empty list then this is immediate. Therefore, assume that $cr_1^* = cr_1^+$, a non-empty list. Furthermore, we can safely assume that the ranges in cr_1^+ are proper. If this is not the case we can first rewrite the ranges using equations [1] and [2]. Equations [34], [35] and [36] merge two range lists that are already sorted into a sorted list. □

Finally, we show that any two character classes are equivalent if and only if they have the same, sorted normal form.

Theorem 1.9 (Correctness) $\forall cc_1, cc_2 : cc_1 \equiv cc_2 \iff \exists cc_3 : cc_1 \twoheadrightarrow cc_3 \twoheadleftarrow cc_2 \wedge \text{sorted}(cc_3)$

Proof. ($\Rightarrow$) According to Lemma 1.8 there are sorted cc_3 and cc_4 such that $cc_1 \twoheadrightarrow cc_3$ and $cc_2 \twoheadrightarrow cc_4$. By Lemma 1.3 we have that $cc_3 \equiv cc_1$ and $cc_4 \equiv cc_2$. We now have $cc_3 \equiv cc_1 \equiv cc_2 \equiv cc_4$ and thus $cc_3 \equiv cc_4$. By sortedness of cc_3 and cc_4 and Lemma 1.6 it then follows that $cc_3 = cc_4$.

($\Leftarrow$) By Lemma 1.3 we have that $cc_1 \equiv cc_3$ and $cc_2 \equiv cc_3$. By transitivity of $\equiv$, $cc_1 \equiv cc_2$. □

B Auxiliary Modules

B.1 Booleans

module Booleans
imports Layout
exports
  sorts Bool
  context-free syntax
    "⊤" → Bool
    "⊥" → Bool
    "∼" Bool → Bool

Bool "$\wedge$" Bool $\rightarrow$ Bool {**assoc**}
Bool "$\oplus$" Bool $\rightarrow$ Bool {**assoc**}
Bool "$\vee$" Bool $\rightarrow$ Bool {**assoc**}
"(" Bool ")" $\rightarrow$ Bool {**bracket**}
"if" Bool "then" Bool "else" Bool "fi" $\rightarrow$ Bool

priorities

"$\sim$"Bool $\rightarrow$ Bool $>$ Bool "$\wedge$"Bool $\rightarrow$ Bool $>$ Bool "$\oplus$"Bool $\rightarrow$ Bool $>$ Bool "$\vee$"Bool $\rightarrow$ Bool

variables

"*Bool*"[*0-9'*]* $\rightarrow$ Bool

equations

[1] $\sim \bot = \top$
[2] $\sim \top = \bot$
[3] $\top \wedge Bool = Bool$
[4] $\bot \wedge Bool = \bot$
[5] $\top \oplus Bool = \sim Bool$
[6] $\bot \oplus Bool = Bool$
[7] $\top \vee Bool = \top$
[8] $\bot \vee Bool = Bool$
[9] if $\top$ then $Bool_1$ else $Bool_2$ fi $= Bool_1$
[10] if $\bot$ then $Bool_1$ else $Bool_2$ fi $= Bool_2$

B.2 Basic Integers

module Basic-Integers
imports IntCon
exports

sorts Int

context-free syntax

IntCon $\rightarrow$ Int
con(Int) $\rightarrow$ IntCon
abs(Int) $\rightarrow$ NatCon
Int "+" Int $\rightarrow$ Int {**left**}

hiddens

variables

"*c*"[*0-9*]* $\rightarrow$ CHAR
"*c**"[*0-9*]* $\rightarrow$ CHAR*
"*c+*"[*0-9*]* $\rightarrow$ CHAR+

equations

Constant

[1] $\text{con}(z) = z$

Absolute value

[2] $\text{abs}(n) = n$
[3] $\text{abs}(-n) = n$

Addition

[4] $$0 + n = n$$
[5] $$n + 0 = n$$

Addition of single digit natural numbers

[6]	$1 + 1 = 2$	[7]	$1 + 2 = 3$	[8]	$1 + 3 = 4$
[9]	$1 + 4 = 5$	[10]	$1 + 5 = 6$	[11]	$1 + 6 = 7$
[12]	$1 + 7 = 8$	[13]	$1 + 8 = 9$	[14]	$1 + 9 = 10$
[15]	$2 + 1 = 3$	[16]	$2 + 2 = 4$	[17]	$2 + 3 = 5$
[18]	$2 + 4 = 6$	[19]	$2 + 5 = 7$	[20]	$2 + 6 = 8$
[21]	$2 + 7 = 9$	[22]	$2 + 8 = 10$	[23]	$2 + 9 = 11$
[24]	$3 + 1 = 4$	[25]	$3 + 2 = 5$	[26]	$3 + 3 = 6$
[27]	$3 + 4 = 7$	[28]	$3 + 5 = 8$	[29]	$3 + 6 = 9$
[30]	$3 + 7 = 10$	[31]	$3 + 8 = 11$	[32]	$3 + 9 = 12$
[33]	$4 + 1 = 5$	[34]	$4 + 2 = 6$	[35]	$4 + 3 = 7$
[36]	$4 + 4 = 8$	[37]	$4 + 5 = 9$	[38]	$4 + 6 = 10$
[39]	$4 + 7 = 11$	[40]	$4 + 8 = 12$	[41]	$4 + 9 = 13$
[42]	$5 + 1 = 6$	[43]	$5 + 2 = 7$	[44]	$5 + 3 = 8$
[45]	$5 + 4 = 9$	[46]	$5 + 5 = 10$	[47]	$5 + 6 = 11$
[48]	$5 + 7 = 12$	[49]	$5 + 8 = 13$	[50]	$5 + 9 = 14$
[51]	$6 + 1 = 7$	[52]	$6 + 2 = 8$	[53]	$6 + 3 = 9$
[54]	$6 + 4 = 10$	[55]	$6 + 5 = 11$	[56]	$6 + 6 = 12$
[57]	$6 + 7 = 13$	[58]	$6 + 8 = 14$	[59]	$6 + 9 = 15$
[60]	$7 + 1 = 8$	[61]	$7 + 2 = 9$	[62]	$7 + 3 = 10$
[63]	$7 + 4 = 11$	[64]	$7 + 5 = 12$	[65]	$7 + 6 = 13$
[66]	$7 + 7 = 14$	[67]	$7 + 8 = 15$	[68]	$7 + 9 = 16$
[69]	$8 + 1 = 9$	[70]	$8 + 2 = 10$	[71]	$8 + 3 = 11$
[72]	$8 + 4 = 12$	[73]	$8 + 5 = 13$	[74]	$8 + 6 = 14$
[75]	$8 + 7 = 15$	[76]	$8 + 8 = 16$	[77]	$8 + 9 = 17$
[78]	$9 + 1 = 10$	[79]	$9 + 2 = 11$	[80]	$9 + 3 = 12$
[81]	$9 + 4 = 13$	[82]	$9 + 5 = 14$	[83]	$9 + 6 = 15$
[84]	$9 + 7 = 16$	[85]	$9 + 8 = 17$	[86]	$9 + 9 = 18$

Addition of multiple digit natural numbers

[87] $$\frac{\begin{array}{c}\mathrm{natcon}(c_1) + \mathrm{natcon}(c_2) = \mathrm{natcon}(c^*\ c),\\ \mathrm{natcon}(c_1^+) + \mathrm{natcon}(c_2^+) + \mathrm{natcon}(\texttt{"0"}\ c^*) = \mathrm{natcon}(c^+)\end{array}}{\mathrm{natcon}(c_1^+\ c_1) + \mathrm{natcon}(c_2^+\ c_2)\ =\ \mathrm{natcon}(c^+\ c)}$$

[88] $$\frac{\begin{array}{c}\mathrm{natcon}(c_1) + \mathrm{natcon}(c_2) = \mathrm{natcon}(c^*\ c),\\ \mathrm{natcon}(\texttt{"0"}\ c_1^*) + \mathrm{natcon}(c_2^+) + \mathrm{natcon}(\texttt{"0"}\ c^*) = \mathrm{natcon}(c^+)\end{array}}{\mathrm{natcon}(c_1^*\ c_1) + \mathrm{natcon}(c_2^+\ c_2)\ =\ \mathrm{natcon}(c^+\ c)}$$

[89] $$\frac{\begin{array}{c}\mathrm{natcon}(c_1) + \mathrm{natcon}(c_2) = \mathrm{natcon}(c^*\ c),\\ \mathrm{natcon}(c_1^+) + \mathrm{natcon}(\texttt{"0"}\ c_2^*) + \mathrm{natcon}(\texttt{"0"}\ c^*) = \mathrm{natcon}(c^+)\end{array}}{\mathrm{natcon}(c_1^+\ c_1) + \mathrm{natcon}(c_2^*\ c_2)\ =\ \mathrm{natcon}(c^+\ c)}$$

B.3 Integers

module Integers
imports Basic-Integers$^{B.2}$ Booleans$^{B.1}$
exports
context-free syntax
Int "−" Int → Int {**left**}
Int "∗" Int → Int {**left**}
"(" Int ")" → Int {**bracket**}

Int ">" Int → Bool
Int "≥" Int → Bool
Int "<" Int → Bool
Int "≤" Int → Bool
priorities
Int "∗"Int → Int > {**left:** Int "+"Int → Int, Int "−"Int → Int}

hiddens
context-free syntax
NatCon "−/" NatCon → NatCon
NatCon "−//" NatCon → NatCon
NatCon "≻" NatCon → NatCon
"(" NatCon ")" → NatCon {**bracket**}
gt "(" NatCon "," NatCon ")" → Bool
variables
"c"[*0-9*]∗ → CHAR
"c∗"[*0-9*]∗ → CHAR∗
"c+"[*0-9*]∗ → CHAR+
equations
Cut off subtraction

[1] $\text{natcon}(c) \mathbin{-\!\!/\!/} \text{natcon}(c) = 0$
[2] $\text{natcon}(c) \mathbin{-\!\!/\!/} 0 = \text{natcon}(c)$

Cut off subtraction for single digit natural numbers

[3]	$2 \mathbin{-\!\!/\!/} 1 = 1$							
[4]	$3 \mathbin{-\!\!/\!/} 1 = 2$	[5]	$3 \mathbin{-\!\!/\!/} 2 = 1$					
[6]	$4 \mathbin{-\!\!/\!/} 1 = 3$	[7]	$4 \mathbin{-\!\!/\!/} 2 = 2$	[8]	$4 \mathbin{-\!\!/\!/} 3 = 1$			
[9]	$5 \mathbin{-\!\!/\!/} 1 = 4$	[10]	$5 \mathbin{-\!\!/\!/} 2 = 3$	[11]	$5 \mathbin{-\!\!/\!/} 3 = 2$			
[12]	$5 \mathbin{-\!\!/\!/} 4 = 1$							
[13]	$6 \mathbin{-\!\!/\!/} 1 = 5$	[14]	$6 \mathbin{-\!\!/\!/} 2 = 4$	[15]	$6 \mathbin{-\!\!/\!/} 3 = 3$			
[16]	$6 \mathbin{-\!\!/\!/} 4 = 2$	[17]	$6 \mathbin{-\!\!/\!/} 5 = 1$					
[18]	$7 \mathbin{-\!\!/\!/} 1 = 6$	[19]	$7 \mathbin{-\!\!/\!/} 2 = 5$	[20]	$7 \mathbin{-\!\!/\!/} 3 = 4$			
[21]	$7 \mathbin{-\!\!/\!/} 4 = 3$	[22]	$7 \mathbin{-\!\!/\!/} 5 = 2$	[23]	$7 \mathbin{-\!\!/\!/} 6 = 1$			
[24]	$8 \mathbin{-\!\!/\!/} 1 = 7$	[25]	$8 \mathbin{-\!\!/\!/} 2 = 6$	[26]	$8 \mathbin{-\!\!/\!/} 3 = 5$			
[27]	$8 \mathbin{-\!\!/\!/} 4 = 4$	[28]	$8 \mathbin{-\!\!/\!/} 5 = 3$	[29]	$8 \mathbin{-\!\!/\!/} 6 = 2$			
[30]	$8 \mathbin{-\!\!/\!/} 7 = 1$							
[31]	$9 \mathbin{-\!\!/\!/} 1 = 8$	[32]	$9 \mathbin{-\!\!/\!/} 2 = 7$	[33]	$9 \mathbin{-\!\!/\!/} 3 = 6$			
[34]	$9 \mathbin{-\!\!/\!/} 4 = 5$	[35]	$9 \mathbin{-\!\!/\!/} 5 = 4$	[36]	$9 \mathbin{-\!\!/\!/} 6 = 3$			
[37]	$9 \mathbin{-\!\!/\!/} 7 = 2$	[38]	$9 \mathbin{-\!\!/\!/} 8 = 1$					

[39] $10 -// 1 = 9$ [40] $10 -// 2 = 8$ [41] $10 -// 3 = 7$
[42] $10 -// 4 = 6$ [43] $10 -// 5 = 5$ [44] $10 -// 6 = 4$
[45] $10 -// 7 = 3$ [46] $10 -// 8 = 2$ [47] $10 -// 9 = 1$

Greater than

[48] $$\frac{\text{natcon}(c_1) -// \text{natcon}(c_2) = \text{natcon}(c),\ \text{natcon}(c) \neq 0}{\text{gt}(\text{natcon}(c_1), \text{natcon}(c_2)) = \top}$$

[49] $$\text{gt}(\text{natcon}(c_1^+\ c_1), \text{natcon}(c_2)) = \top$$

[50] $$\text{gt}(\text{natcon}(c^+\ c_1), \text{natcon}(c^+\ c_2)) = \text{gt}(\text{natcon}(c_1), \text{natcon}(c_2))$$

[51] $$\frac{\text{gt}(\text{natcon}(c_1^+), \text{natcon}(c_2^+)) = \top}{\text{gt}(\text{natcon}(c_1^+\ c_1), \text{natcon}(c_2^+\ c_2)) = \top}$$

Partial substraction

[52] $$n \succ 0 = n$$

[53] $$\text{natcon}(c_1) \succ \text{natcon}(c_2) = \text{natcon}(c_1) -// \text{natcon}(c_2)$$

[54] $$\frac{\begin{array}{c}\text{natcon}(c_1) -// \text{natcon}(c_2) = \text{natcon}(c_3),\\ \text{natcon}(c_1^+) \succ \text{natcon}(\texttt{"0"}\ c_2^*) = \text{natcon}(c^+)\end{array}}{\text{natcon}(c_1^+\ c_1) \succ \text{natcon}(c_2^*\ c_2) = \text{natcon}(c^+\ c_3)}$$

[55] $$\frac{\begin{array}{c}\text{natcon}(c_2) -// \text{natcon}(c_1) = \text{natcon}(c_3),\\ 10 -// \text{natcon}(c_3) = \text{natcon}(c),\\ \text{natcon}(\texttt{"0"}\ c_2^*) + 1 = n,\\ \text{natcon}(c_1^+) \succ n = \text{natcon}(c^+)\end{array}}{\text{natcon}(c_1^+\ c_1) \succ \text{natcon}(c_2^*\ c_2) = \text{natcon}(c^+\ c)}$$

Cut off subtraction -/

[56] $$n_1 -/ n_2 = n_1 \succ n_2 \quad \textbf{when}\ \text{gt}(n_1, n_2) = \top$$
[57] $$n_1 -/ n_2 = 0 \quad \textbf{when}\ \text{gt}(n_1, n_2) \neq \top$$

Subtraction of naturals

[58] $$n_1 - n_2 = n_1 \succ n_2 \quad \textbf{when}\ \text{gt}(n_1, n_2) = \top$$
[59] $$n_1 - n_2 = - n_2 \succ n_1 \quad \textbf{when}\ \text{gt}(n_1, n_2) \neq \top$$

Multiplication of naturals

[60] $$n * 0 = 0$$

[61] $$n * 1 = n$$

[62] $$\frac{\text{gt}(\text{natcon}(c), 1) = \top}{n * \text{natcon}(c) = n + n * (\text{natcon}(c) - 1)}$$

[63] $$\frac{\text{natcon}(c_1^+) * \text{natcon}(c_2^+) = \text{natcon}(c^+)}{\text{natcon}(c_1^+) * \text{natcon}(c_2^+\ c) = \text{natcon}(c^+\ \texttt{"0"}) + \text{natcon}(c_1^+) * \text{natcon}(c)}$$

Addition, subtraction, and multiplication of integers

[64] $n_1 + - n_2 = n_1 - n_2$

[65] $- n_1 + n_2 = n_2 - n_1$

[66] $- n_1 + - n_2 = - n \quad \textbf{when} \;\; n_1 + n_2 = n$

[67] $n_1 - - n_2 = n_1 + n_2$

[68] $- n_1 - n_2 = - n \quad \textbf{when} \;\; n_1 + n_2 = n$

[69] $- n_1 - - n_2 = n_2 - n_1$

[70] $n_1 * - n_2 = - n \quad \textbf{when} \;\; n_1 * n_2 = n$

[71] $- n_1 * n_2 = - n \quad \textbf{when} \;\; n_1 * n_2 = n$

[72] $- n_1 * - n_2 = n_1 * n_2$

Relational operators

[73] $n_1 > n_2 = \top \quad \textbf{when} \;\; \mathrm{gt}(n_1, n_2) = \top$

[74] $n_1 > n_2 = \bot \quad \textbf{when} \;\; \mathrm{gt}(n_1, n_2) \neq \top$

[75] $n_1 > - n_2 = \top$

[76] $- n_1 > n_2 = \bot$

[77] $- n_1 > - n_2 = n_2 > n_1$

[78] $z_1 \geq z_2 = z_1 > z_2 \quad \textbf{when} \;\; z_1 \neq z_2$

[79] $z \geq z = \top$

[80] $z_1 < z_2 = \sim z_1 \geq z_2$

[81] $z_1 \leq z_2 = \sim z_1 > z_2$

Technical Reports of the Programming Research Group

Note: These reports can be obtained using the technical reports overview on our WWW site (URL `http://www.wins.uva.nl/research/prog/reports/`) or using anonymous ftp to `ftp.wins.uva.nl`, directory `pub/programming-research/reports/`.

[P9711] L. Moonen. *A Generic Architecture for Data Flow Analysis to Support Reverse Engineering.*

[P9710] B. Luttik and E. Visser. *Specification of Rewriting Strategies.*

[P9709] J.A. Bergstra and M.P.A. Sellink. *An Arithmetical Module for Rationals and Reals.*

[P9708] E. Visser. *Character Classes.*

[P9707] E. Visser. *Scannerless generalized-LR parsing.*

[P9706] E. Visser. *A family of syntax definition formalisms.*

[P9705] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Generation of Components for Software Renovation Factories from Context-free Grammars.*

[P9704] P.A. Olivier. *Debugging Distributed Applications Using a Coordination Architecture.*

[P9703] H.P. Korver and M.P.A. Sellink. *A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes.*

[P9702] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Reengineering COBOL Software Implies Specification of the Underlying Dialects.*

[P9701] E. Visser. *Polymorphic Syntax Definition.*

[P9618] M.G.J. van den Brand, P. Klint, and C. verhoef. *Re-engineering needs Generic Programming Language Technology.*

[P9617] P.I. Manuel. *ANSI Cobol III in SDF + an ASF Definition of a Y2K Tool.*

[P9616] P.H. Rodenburg. *A Complete System of Four-valued Logic.*

[P9615] S.P. Luttik and P.H. Rodenburg. *Transformations of Reduction Systems.*

[P9614] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Core Technologies for System Renovation.*

[P9613] L. Moonen. *Data Flow Analysis for Reverse Engineering.*

[P9612] J.A. Hillebrand. *Transforming an ASF+SDF Specification into a ToolBus Application.*

[P9611] M.P.A. Sellink. *On the conservativity of Leibniz Equality.*

[P9610] T.B. Dinesh and S.M. Üsküdarlı. *Specifying input and output of visual languages.*

[P9609] T.B. Dinesh and S.M. Üsküdarlı. *The VAS formalism in VASE.*

[P9608] J.A. Hillebrand. *A small language for the specification of Grid Protocols.*

[P9607] J.J. Brunekreef. *A transformation tool for pure Prolog programs: the algebraic specification.*

[P9606] E. Visser. *Solving type equations in multi-level specifications (preliminary version).*

[P9605] P.R. D'Argenio and C. Verhoef. *A general conservative extension theorem in process algebras with inequalities.*

[P9602b] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives (revised version of P9602).*

[P9604] E. Visser. *Multi-level specifications.*

[P9603] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Reverse engineering and system renovation: an annotated bibliography.*

[P9602] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives.*

[P9601] P.A. Olivier. *Embedded system simulation: testdriving the ToolBus.*

[P9512] J.J. Brunekreef. *TransLog, an interactive tool for transformation of logic programs.*

[P9511] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. *Grid protocols based on synchronous communication: specification and correctness.*

[P9510] P.H. Rodenburg. *Termination and confluence in infinitary term rewriting.*

[P9509] J.A. Bergstra and Gh. Stefanescu. *Network algebra with demonic relation operators.*

[P9508] J.A. Bergstra, C.A. Middelburg, and Gh. Stefanescu. *Network algebra for synchronous and asynchronous dataflow.*

[P9507] E. Visser. *A case study in optimizing parsing schemata by disambiguation filters.*

[P9506] M.G.J. van den Brand and E. Visser. *Generation of formatters for context-free languages.*

[P9505] J.M.T. Romijn. *Automatic analysis of term rewriting systems: proving properties of term rewriting systems derived from* ASF+SDF *specifications.*