

$(\forall\sigma\alpha\Theta\exists\sigma\alpha\Pi)$

sTs

HOL Extended with MF-ZFC Sets

(The Mostly First-Order Language of ZFC Sets)

Gottfried Barrow

v.2013-02-27.03:28:24

# Short Contents

## Title

<b>Short Contents</b>	<b>i</b>
<b>Table of Contents</b>	<b>ii</b>
<b>1 Preliminary Prerequisite Knowledge, TODOs</b>	<b>1</b>
1.1 Theory Begin . . . . .	1
1.2 TODO: Brief Tutorials . . . . .	1
<b>2 Notational and Operator Overhead</b>	<b>2</b>
2.1 Guidelines: Naming, Notation, and Fonts . . . . .	2
2.2 A Needed Operator . . . . .	3
2.3 Variable Naming Convention . . . . .	4
<b>3 Existence, Extension, Unordered Pairs</b>	<b>5</b>
3.1 Axiom of Extension (Set Equality Axiom) . . . . .	5
3.2 Axiom of Existence (Empty Set Axiom) . . . . .	8
3.3 Axiom of Pairs . . . . .	10
3.4 Ordered Pairs . . . . .	13
<b>4 Separation, Union, Power Set</b>	<b>17</b>
4.1 Axiom Schema of Separation . . . . .	17
4.2 Axiom of Unions . . . . .	21
4.3 Finite Sets . . . . .	32
4.4 Intersection . . . . .	36
4.5 Axiom of Power Sets . . . . .	40
<b>5 ***** WORKING HERE START *****</b>	<b>42</b>
5.1 ***** WORKING START ***** . . . . .	42
5.2 ***** WORKING END ***** . . . . .	42
<b>6 ***** WORKING HERE END *****</b>	<b>43</b>
<b>Bibliography</b>	<b>44</b>
<b>Index</b>	<b>45</b>

# Contents

## Title

## Short Contents

i

## Table of Contents

ii

<b>1 Preliminary Prerequisite Knowledge, TODOs</b>	<b>1</b>
1.1 Theory Begin . . . . .	1
....Is.1.1.1.....[Theory name, imports, and begin] . . . . .	1
1.2 TODO: Brief Tutorials . . . . .	1
<b>2 Notational and Operator Overhead</b>	<b>2</b>
2.1 Guidelines: Naming, Notation, and Fonts . . . . .	2
2.1.1 (Constant and Function Naming) . . . . .	2
2.1.2 (Typewriter Font, Math Font, and Isar Inside Math) . . . . .	2
2.1.3 (Isar Syntax Theorem Naming) . . . . .	3
2.2 A Needed Operator . . . . .	3
....Nn.2.2.1.....[impliedby operator: long left arrow] . . . . .	3
2.3 Variable Naming Convention . . . . .	4
<b>3 Existence, Extension, Unordered Pairs</b>	<b>5</b>
3.1 Axiom of Extension (Set Equality Axiom) . . . . .	5
3.1.1 (The Primitive Set Type and Membership Predicate) . . . . .	5
....Ty.3.1.2.....[sT primitive set type: everything is a set] . . . . .	5
....Ty.3.1.3.....[sTL::(sT list), bT::bool] . . . . .	5
....Pd.3.1.4.....[inP membership: axiomatized by subsequent axioms] . . . . .	5
3.1.5 (The Axiom of Extension) . . . . .	5
....Ax.3.1.6.....[Axiom of Extension: set equality] . . . . .	6
3.1.7 (Subsets) . . . . .	6
....Df.3.1.8.....[ss0 subset, su0 superset, ns0 not a subset] . . . . .	6
....Nn.3.1.9.....[ssf0, suf0, and nsf0 subset operator formulas] . . . . .	6
....Nn.3.1.10.....[es0 equal subsets] . . . . .	7
....Th.3.1.11.....[ss0 equals ssf0] . . . . .	7
....Th.3.1.12.....[ss0 equality] . . . . .	8
....Th.3.1.13.....[ss0 is reflexive] . . . . .	8
....Th.3.1.14.....[ss0 is transitive] . . . . .	8
3.2 Axiom of Existence (Empty Set Axiom) . . . . .	8
3.2.1 (Empty Set Constant, Axiom of Existence) . . . . .	8
....Cx.3.2.2.....[emS empty set: axiomatized by the Axiom of Existence] . . . . .	9
....Ax.3.2.3.....[Axiom of Existence: the empty set contains no elements] . . . . .	9
....Ax.3.2.4.....[Axiom of Existence: no constant form] . . . . .	9
3.2.5 (Empty Set Uniqueness, emS Is a Subset of Every Set) . . . . .	9
....Th.3.2.6.....[Empty set uniqueness] . . . . .	9
....Th.3.2.7.....[emS is unique] . . . . .	9
....Th.3.2.8.....[emS ss0 r] . . . . .	10

3.3	Axiom of Pairs . . . . .	10
3.3.1	(Unordered Pairs, Axiom, Uniqueness, Existence) . . . . .	10
....Cx.3.3.2.....	[paS unordered pair: axiomatized by the Axiom of Pairs] . . . . .	10
....Ax.3.3.3.....	[Axiom of Pairs: unordered pairs exist] . . . . .	10
....Ax.3.3.4.....	[Axiom of Pairs: no constant form] . . . . .	10
....Th.3.3.5.....	[paS is unique] . . . . .	11
3.3.6	(Singleton Existence) . . . . .	11
....Df.3.3.7.....	[siS singleton] . . . . .	11
....Th.3.3.8.....	[Singletons exist] . . . . .	11
....Th.3.3.9.....	[siS exists] . . . . .	12
....Th.3.3.10.....	[Singleton uniqueness] . . . . .	12
....Th.3.3.11.....	[siS is unique] . . . . .	12
....Th.3.3.12.....	[siS is a pair] . . . . .	12
....Th.3.3.13.....	[emS equals {r} is false] . . . . .	12
3.3.14	(Unordered Pairs Are Unordered, Element Matching) . . . . .	13
....Th.3.3.15.....	[paS is unordered] . . . . .	13
....Co.3.3.16.....	[paS {r, s} {p} = paS {p} {r, s}] . . . . .	13
....Th.3.3.17.....	[paS element match] . . . . .	13
3.4	Ordered Pairs . . . . .	13
3.4.1	(Ordered Pair Definition, Notation, Existence) . . . . .	13
....Nn.3.4.2.....	[opS ordered pair] . . . . .	13
....Th.3.4.3.....	[Ordered pairs exist] . . . . .	14
....Th.3.4.4.....	[opS exists] . . . . .	14
3.4.5	(Ordered Pair Uniqueness) . . . . .	14
....Th.3.4.6.....	[Ordered pair uniqueness] . . . . .	14
....Th.3.4.7.....	[opS is unique] . . . . .	14
3.4.8	(Ordered Pair Element Matching) . . . . .	15
....Th.3.4.9.....	[opS element match] . . . . .	15
3.4.10	(Using Ordered Pairs to Recursively Define n-tuples) . . . . .	16
<b>4</b>	<b>Separation, Union, Power Set</b> . . . . .	<b>17</b>
4.1	Axiom Schema of Separation . . . . .	17
4.1.1	(Separation Set Constant, Notation, and Axiom) . . . . .	17
....Cx.4.1.2.....	[seS separation: axiomatized by the Axiom of Separation] . . . . .	17
....Nn.4.1.3.....	[Set builder notation: all $x$ in $q$ such that $P$ ] . . . . .	17
....Ax.4.1.4.....	[Axiom of Separation: separation sets] . . . . .	17
....Ax.4.1.5.....	[Axiom of Separation: no constant form] . . . . .	17
4.1.6	(Separation Set Builder Notation) . . . . .	18
....Th.4.1.7.....	[A lambda calculus equivalency useful for separation sets] . . . . .	18
....Th.4.1.8.....	[seS equivalent notation] . . . . .	18
4.1.9	(Separation Set Existence and Uniqueness) . . . . .	19
....Th.4.1.10.....	[Separation set uniqueness] . . . . .	19
....Th.4.1.11.....	[seS is unique] . . . . .	19
4.1.12	(Basic Examples Using Set Builder Notation) . . . . .	19
....Xp.4.1.13.....	[emS equals all $x$ not equal to $x$ ] . . . . .	19

...Xp.4.1.14.....[The set $p$ is not in the set not containing $p$ ] . . . . .	19
...Xp.4.1.15.....[Singletons are membership equal to their seS] . . . . .	19
...Xp.4.1.16.....[Singletons equal their seS] . . . . .	20
...Xp.4.1.17.....[The set containing emS equals its seS] . . . . .	20
4.1.18 (An Example of No Free Variable in P) . . . . .	20
...Xp.4.1.19.....[When no free $x$ is in $P_x$ for set builder notation] . . . . .	20
4.2 Axiom of Unions . . . . .	21
4.2.1 (General Union Set Constant and Axiom) . . . . .	21
...Cx.4.2.2.....[geU general union: axiomatized by the Axiom of Unions] . . . . .	21
...Ax.4.2.3.....[Axiom of Unions] . . . . .	21
...Ax.4.2.4.....[Axiom of Unions: no constant form] . . . . .	21
4.2.5 (Coordinating simp Rules for geU) . . . . .	21
4.2.6 (geU Simplification Rules and fiS) . . . . .	22
4.2.7 (Union Uniqueness) . . . . .	22
...Th.4.2.8.....[Union uniqueness] . . . . .	22
...Th.4.2.9.....[geU is unique] . . . . .	23
4.2.10 (paS Unions Exist, inP Or, geU{r} = r) . . . . .	23
...Th.4.2.11.....[Unions of paS exist] . . . . .	23
...Th.4.2.12.....[geU paS exists] . . . . .	24
...Th.4.2.13.....[geU siS equals inP] . . . . .	24
...Th.4.2.14.....[geU paS equals inP or] . . . . .	24
...Th.4.2.15.....[geU{r} = r] . . . . .	24
4.2.16 (Distribute, geU Permutative Rewrite Rules) . . . . .	25
...Th.4.2.17.....[geU p into geU paS] . . . . .	25
...Th.4.2.18.....[geU is associative] . . . . .	25
...Th.4.2.19.....[geU left commute] . . . . .	26
4.2.20 (Pseudo Associate and Commute) . . . . .	26
...Th.4.2.21.....[geU{siS, paS} left commute] . . . . .	26
...Th.4.2.22.....[geU{paS, paS} inside commute] . . . . .	26
...Th.4.2.23.....[geU{paS, siS} associates] . . . . .	27
4.2.24 (Reducing paS to siS, Eliminating emS) . . . . .	27
...Th.4.2.25.....[geU{paS, paS} twin isolate] . . . . .	27
...Th.4.2.26.....[geU emS = emS] . . . . .	27
...Th.4.2.27.....[geU{emS, r} = r] . . . . .	28
4.2.28 (Eliminating All Unions) . . . . .	28
...Th.4.2.29.....[geU{siS, siS} = paS] . . . . .	28
...Th.4.2.30.....[geU{{r}, {r, s}} = {r, s}] . . . . .	28
4.2.31 (Eliminating a Union of Union, Ordering to the Right) . . . . .	28
...Th.4.2.32.....[geU geU{r, {s}} = geU{s, geU r}] . . . . .	28
...Co.4.2.33.....[geU geU{{r}, s} = geU{r, geU s}] . . . . .	29
...Th.4.2.34.....[geU{{p, r}, s} = geU{{p}, geU{{r}, s}}] . . . . .	29
4.2.35 (Eliminating Some Unions) . . . . .	29
...Th.4.2.36.....[geU{r, geU{r, s}} = geU{r, s}] . . . . .	29
...Th.4.2.37.....[geU{r, geU{s}} = geU{r, s}] . . . . .	30
...Co.4.2.38.....[geU{geU{r}, geU{s}} = geU{r, s}] . . . . .	30

...Th.4.2.39.....	[geU{geU{r, s}} = geu{r, s}] . . . . .	30
4.2.40	(Miscellaneous Results) . . . . .	30
...Th.4.2.41.....	[p in r implies p a subset of geU r] . . . . .	30
...Th.4.2.42.....	[geU{x, {x}} exists] . . . . .	31
4.2.43	(biU Binary Union) . . . . .	31
...Nn.4.2.44.....	[biU Binary Union] . . . . .	31
...Co.4.2.45.....	[biU commutes] . . . . .	31
...Co.4.2.46.....	[biU left commutes] . . . . .	31
...Co.4.2.47.....	[biU is associative] . . . . .	32
...Co.4.2.48.....	[biU distribute into union] . . . . .	32
4.3	Finite Sets . . . . .	32
4.3.1	(Finite set fiS, Equality Examples) . . . . .	32
...Rd.4.3.2.....	[fiS finite set] . . . . .	32
...Xp.4.3.3.....	[Using geU and paS to build {a, b, c}] . . . . .	32
...Xp.4.3.4.....	[fiS has no order, repetitions do not matter] . . . . .	33
...Xp.4.3.5.....	[geU unions of fiS] . . . . .	33
4.3.6	(Converting to Cons) . . . . .	34
...Th.4.3.7.....	[fiS([r] @ rs) = fiS(r # rs)] . . . . .	34
...Th.4.3.8.....	[fiS(rs @ [r]) = fiS(r # rs)] . . . . .	34
...Th.4.3.9.....	[geU(fiS(r # rs)) = geU{r,geU(fiS rs)}] . . . . .	34
...Th.4.3.10.....	[geU(fiS([r] @ rs)) = geU{r,geU(fiS rs)}] . . . . .	35
4.3.11	(Converting Appends to a Union, Simplifying Unions of fiS) . . . . .	35
...Th.4.3.12.....	[geU{fiS} = fiS] . . . . .	35
...Th.4.3.13.....	[fiS append to pair] . . . . .	35
...Th.4.3.14.....	[geU{geU fiS} = geU fiS] . . . . .	35
...Th.4.3.15.....	[geU{geU fiS,geU fiS} = geU{geU{fiS,fiS}}] . . . . .	35
...Xp.4.3.16.....	[fiS equations solved using simp only] . . . . .	36
4.4	Intersection . . . . .	36
4.4.1	(Definitions, inP And) . . . . .	36
...Df.4.4.2.....	[geI general intersection] . . . . .	36
...Nn.4.4.3.....	[biI binary intersection] . . . . .	36
...Th.4.4.4.....	[geI siS equals inP] . . . . .	37
...Th.4.4.5.....	[geI paS equals inP and] . . . . .	37
4.4.6	(NEW: Permutative Rewrite Rules, geI{r} = r) . . . . .	37
...Th.4.4.7.....	[NEW: geI is associative] . . . . .	37
...Th.4.4.8.....	[NEW: geI left commute] . . . . .	37
...Th.4.4.9.....	[geI{r} = r] . . . . .	38
...Th.4.4.10.....	[NEW: geI of two singletons] . . . . .	38
4.4.11	(NEW: Distribute) . . . . .	38
...Th.4.4.12.....	[NEW: Left distribute] . . . . .	38
...Th.4.4.13.....	[NEW: Right distribute] . . . . .	38
...Th.4.4.14.....	[NEW: Left distribute into pair] . . . . .	39
...Th.4.4.15.....	[NEW: Right distribute into pair] . . . . .	39
4.4.16	(NEW: Example) . . . . .	39
...Xp.4.4.17.....	[NEW: example] . . . . .	39

4.5	Axiom of Power Sets . . . . .	40
4.5.1	(Power Set Constant and Axiom) . . . . .	40
	....Cx.4.5.2.....[pwS power set: axiomatized by the Axiom of Power Sets] . . . . .	40
	....Ax.4.5.3.....[Axiom of Power Sets] . . . . .	40
	....Ax.4.5.4.....[Axiom of Power Sets: no constant form] . . . . .	40
4.5.5	(Power Set Uniqueness) . . . . .	41
	....Th.4.5.6.....[Power set uniqueness] . . . . .	41
	....Th.4.5.7.....[pwS is unique] . . . . .	41
<b>5</b>	<b>***** WORKING HERE START *****</b>	<b>42</b>
5.1	<b>***** WORKING START *****</b> . . . . .	42
5.2	<b>***** WORKING END *****</b> . . . . .	42
<b>6</b>	<b>***** WORKING HERE END *****</b>	<b>43</b>
	....Is.6.0.1.....[Theory end] . . . . .	43
	<b>Bibliography</b>	<b>44</b>
	<b>Index</b>	<b>45</b>

# 1 Preliminary Prerequisite Knowledge, TODOs

## 1.1 Theory Begin

( $\iota\sigma$ ) (ISAR) 1.1.1. (Theory name, imports, and begin)

```
4 | theory sTs  
5 | imports Complex_Main "i"  
6 | begin
```

## 1.2 TODO: Brief Tutorials

- Brief explanation of the three first-order logic definitions in [Bil03].
- Brief explanation of what `List.list.Cons` and `List.append` are as functional programming concepts, and how to use them in a very basic way for recursion.
- Brief explanation of how to use `using[[simp_trace]]` to discover the details of proofs that use the automatic proof methods `simp` and `auto`.
- How to use `(input)` with `notation` commands to be able to see more less notational detail.



## 2 Notational and Operator Overhead

### 2.1 Guidelines: Naming, Notation, and Fonts

#### 2.1.1 (Constant and Function Naming)

The following list spells out some naming conventions for type and function constants, and related notation and abbreviations:

- The main constants are primarily named using a capital letter with a suffix which is a subscripted capital letter, followed by a subscripted letter. Example:  $P_{In}$ .
- An ASCII form will be provided which will change the suffix to the capital letter into a prefix, where both prefix letters are lowercase. Example: `inP`.
- The order and use of lowercase and uppercase letters is to try and prevent global name clashes with the identifiers that are used in the HOL logic.
- For many functions and constants, the capital letter will be chosen to correspond with standardized, mathematical usage and naming. For example, the “U” in `nfU` and `fiU` has been chosen to correspond with “union”.
- A capital “S” in a name, such as  $S_{Em}$ , is a constant that represents a set.
- A capital “P” in a name, such as  $P_{In}$ , is a function that is a predicate.
- A capital “O” in a name, such as  $O_{Ss}$ , is a function that is an operator.
- A capital “T” in a name, such as  $sT$ , is a type.

Additionally, there may be two or more notations for one constant, so these additional naming conventions are used:

- For a binary operator constant, the name used to define the constant, such as  $P_{In}$ , will be the function application form of the operator, and its ASCII name, such as `inP`, will be the *infix* form of the operator.
- For a binary operator, an ASCII function form will also be defined by inserting an underscore before the capital letter. For example, `in.P` is the function form of `inP`.

#### 2.1.2 (Typewriter Font, Math Font, and Isar Inside Math)

- **Function names:** In text that is not in a math environment, use typewriter font for Isar keywords, identifiers such as theorem names, and for both the ASCII and subscripted name of functions, such as `fiU` and  $U_{Fi}$ .
- **References to syntax:** Use typewriter font when referencing an Isar statement which is inside a verbatim environment. If the syntax also falls the category of being a math expression, then the choice between using typewriter font or a math environment will be whether the emphasis should be placed on the syntax or on the math concept.
- **Math expressions:** In text, use inline math or a math environment for what would be considered a mathematical expression.
- **Isar in math:** In a math environment, for Isar identifiers which are not simple variables, and for function application, use the markup command that has been designated for text inside of a math environment.

- Inside a math environment, if text is not inside a LaTeX command such as `\text`, the text will get italicized, and spaces will get removed.
- Typewriter font does not look completely right when used inside a math environment, and using a different markup command allows the general text font for Isar inside a math environment to be changed.
- There is a conflict between the need to preserve spaces in a functional programming expression, a need for the fonts of all the variables to have a consistent look, and a need to not have to use lots of LaTeX commands inside a math environment to get the Isar syntax to mix well with the standard look of LaTeX mathematics.
- For example, there is  $(\lambda x. P x)$  with `\textttt`,  $(\lambda x. P x)$  with `\text`, and  $(\lambda.Px)$  with `\ensuremath`. The typewriter font will be too big and square when mixed in with italicized variables, and the monospaced font makes the spacing wrong for that expression. In the math environment, the important spacing is lost and the variables are italicized. However, if `\text` is used, and the variables are also being used with a quantifier, such as in  $\forall P.\forall x.(\lambda x. P x)$ , then two different fonts are being used to represent the same variable, yet you definitely do not want to be micromanaging the fonts of variables inside of a math environment.
- For an expression like  $\forall P.\forall x.(\lambda x. P x)$ , there is the option of not using `\text`, and instead micromanaging spacing, such as shown here:  $\forall P.\forall x.(\lambda x. P x)$ . Because variables such as  $x$  and  $P$  have to be discussed in text alongside expressions such as  $\forall P.\forall x.(\lambda x. P x)$ , and because variables need to be italicized when used alone, and because there needs to be some consistency, then the second option seems to be a better choice as a standard rule.
- To summarize, identifiers which are multi-character, such as function constant  $U_{Fi}$ , that are used inside a math environment, should be used with a roman font; single variables or single subscripted variables should be left in the normal math font.

### 2.1.3 (Isar Syntax Theorem Naming)

- A subscripted single quote (') represents a space.
- A subscripted underscore (.) represents a left or right brace.
- A non-subscripted single quote (') represents a left or right bracket for grouping, or a left or right bracket for a list.
- A superscripted underscore (.) represents a comma.
- Operators such as EQ and IFF will be separated on both sides by two of either subscripted single quote or subscripted underscore.

## 2.2 A Needed Operator

It is convenient to use Nitpick to test biconditionals, and when a biconditional is found to be false, Nitpick can then be used to test the two directions of the biconditional.

The  $\longleftrightarrow$  operator can be replaced with  $\longrightarrow$  to test the left to right direction, and to make it easy to test the right to left direction, the abbreviation `impliedby` is defined to give us the  $\longleftarrow$  operator, and its ASCII equivalent `<--`.

The `(input)` used in the abbreviation prevents  $x \longrightarrow y$  from being replaced by  $y \longleftarrow x$ .

$(\forall \forall)$  (NOTATION) 2.2.1. (`impliedby` operator: long left arrow)

```

133 | abbreviation (input)
134 |   "impliedby x y == y → x"
135 |
136 | notation
137 |   impliedby (infixl "<--" 25) and
138 |   impliedby (infixl "←" 25)

```

## 2.3 Variable Naming Convention

The following list describes the preferred naming convention for variables.

- Everything is a set, but some sets are viewed primarily as “sets”, and some are primarily viewed as “elements”.
- For the beginning parts of axiomatic-modeled set theory, lower case is used.
- This is to emphasize first-order formulas. At some point, there is a switch to uppercase to represent “sets”, with lowercase representing “elements of sets”.
- The property variable
  - $P$  is used to represent a property, so  $p$  is not used in a formula in which  $P$  is used.
- Outside universal quantified variables or free variables. Because  $p$  and  $q$  look similar, preference is given to  $r$  and  $s$ .
  - The letters  $p, q, r, s$  are used.
- Inside universal quantified variables thought of as elements.
  - The letters  $x, y, z, w$  are used.
- Existential quantified variables.
  - The letters  $u, v,$  and  $t$  are used. The letter  $a$  is avoided because it looks like the English article “a”.
- Constants, such as used in `value "{a, b}"`.
  - The letters  $a, b, c, c,$  etc. are used.

Numbered subscripts will be used when there are more variables needed that are available for one of the categories. For example,  $q_1, q_2, r_1,$  and  $r_2$  would be used for four free variables rather than using  $r, s, u,$  and  $v$ .

### 3 Existence, Extension, Unordered Pairs

#### 3.1 Axiom of Extension (Set Equality Axiom)

##### 3.1.1 (The Primitive Set Type and Membership Predicate)

ZFC sets is a first-order language which requires an infinite set of variables, and it generally goes unsaid in the formalization of a first-order language that the variables provided are of a single type.

However, in HOL there are a multitude of variable types, and additionally, we are allowed to define a new type of variable so we can have a variable type that exists in its own domain.

For ZFC sets, `sT` is the primitive set type, and for type `sT`,  $\sigma_i$  has been defined for as non-ASCII notation. The subscripted character used in the  $\sigma_i$  is the Greek letter iota.

$(\tau\iota)$  (TYPE) 3.1.2. (`sT` primitive set type: everything is a set)

```
189 | typedecl sT ("σi")
```

$(\tau\iota)$  (TYPE) 3.1.3. (`sTL::(sT list)`, `bT::bool`)

```
193 | type_synonym sTL = "sT list" ("σΛ")
194 |
195 | type_synonym bT = bool ("βi")
```

ZFC sets is specified to have one predicate, which is membership. The ASCII and non-ASCII infix notation for membership are `PIn` and  $\in_i$ , along with `PInf` for ASCII function notation. For negation of membership, there is `niOp`,  $\notin_i$ , and `niOpi`.

The notation for `PIn` is `x\iisub>\<iota>y`

$(\pi\delta)$  (PREDICATE) 3.1.4. (`inP` membership: axiomatized by subsequent axioms)

```
206 | consts PIn :: "σi ⇒ σi ⇒ βi"
207 |
208 | notation
209 |   PIn ("in'_P") and
210 |   PIn (infix "inP" 51) and
211 |   PIn (infix "εi" 51)
212 |
213 | abbreviation PNi :: "σi ⇒ σi ⇒ βi" where
214 |   "PNi p q == ¬(p εi q)"
215 |
216 | notation
217 |   PNi ("ni'_P") and
218 |   PNi (infix "niP" 51) and
219 |   PNi (infix "εi" 51)
```

##### 3.1.5 (The Axiom of Extension)

Because outermost universal quantification of variables is taken care of by the meta-logic, we might get overly ambitious in trying to get rid of universal quantifiers, and state the Axiom of extension as

$$(x \in_i r \longleftrightarrow x \in_i s) \longleftrightarrow (r = s). \quad (3.1.1)$$

However, this formula would allow two sets to be equal with only one element in common. It is best not to make such basic mistakes when stating axioms, since axioms are very uncritical of your logic.

$(\alpha\xi)$  (AXIOM) 3.1.6. (Axiom of Extension: set equality)

```

235 | axiomatization where
236 |   Ax·xN: "( $\forall x. x \in_l r \longleftrightarrow x \in_l s$ ) = (r = s)"
237 |
238 | theorem
239 |   " $\forall r. \forall s. (\forall x. x \in_l r \longleftrightarrow x \in_l s) = (r = s)$ "
240 |   by(metis Ax·xN)

```

### 3.1.7 (Subsets)

Because the subset operator is frequently used to prove set equality, it is introduced here in connection with the Axiom of Extension.

The subset operator could be defined as an abbreviation, but instead, it is defined as a function so that when `simp_trace` is being used, it can be seen that a choice was made to use the subset operator, rather than its defining formula.

However, there are multiple needs, and another need is to expand the subset operator into its defining formula with a `simp` rule so that logical decisions can be made using  $\in_l$ , for example, to decide whether  $x$  is in  $\{a, b\}$  based on the formula  $(x = a \vee x = b)$ .

The result, though, of converting the subset operator into its defining formula with a `simp` rule is that after that the rule is in place, the subset operator cannot be used on the left-hand side of a subsequent `simp` rule. This is because the simplifier will rewrite the subset operator before any such subsequent rule can be used.

The solution to the multiple needs and complications is to provide  $\subset_l$  as a true operator, and  $\subset_\phi$  as an abbreviation for the subset formula, where  $\subset_\phi$  will be used in theorems which are also `simp` rules.

$(\Delta)$  (DEFINITION) 3.1.8. (ss0 subset, su0 superset, ns0 not a subset)

```

269 | definition 0Ss :: " $\sigma_l \Rightarrow \sigma_l \Rightarrow \beta_l$ " where
270 |   "0Ss r s = ( $\forall x. x \in_l r \longrightarrow x \in_l s$ )"
271 | notation
272 |   0Ss ("ss'_0") and
273 |   0Ss (infix "ss0" 51) and
274 |   0Ss (infix " $\subset_l$ " 51)
275 |
276 | abbreviation (input) 0Su :: " $\sigma_l \Rightarrow \sigma_l \Rightarrow \beta_l$ " where
277 |   "0Su r s == 0Ss s r"
278 | notation
279 |   0Su ("su'_0") and
280 |   0Su (infix "su0" 51) and
281 |   0Su (infix " $\supset_l$ " 51)
282 |
283 | abbreviation 0Ns :: " $\sigma_l \Rightarrow \sigma_l \Rightarrow \beta_l$ " where
284 |   "0Ns r s ==  $\neg(r \subset_l s)$ "
285 | notation
286 |   0Ns ("ns'_0") and
287 |   0Ns (infix "ns0" 51) and
288 |   0Ns (infix " $\sim\subset_l$ " 51)

```

With abbreviation, internally `ssf0` will be replaced by its defining FOL formula, while for the user, instead of the FOL formula being displayed, `ssf0` will be displayed [Nip13, 14].

$(\nu\nu)$  (NOTATION) 3.1.9. (ssf0, suf0, and nsf0 subset operator formulas)

```

296 abbreviation OSSF :: " $\sigma_i \Rightarrow \sigma_i \Rightarrow \beta_i$ " where
297   "OSSF r s == ( $\forall x. x \in_i r \longrightarrow x \in_i s$ )"
298 notation
299   OSSF ("ssf'_0") and
300   OSSF (infix "ssf0" 51) and
301   OSSF (infix " $\subset_\phi$ " 51)
302
303 abbreviation (input) OSUF :: " $\sigma_i \Rightarrow \sigma_i \Rightarrow \beta_i$ " where
304   "OSUF r s == OSSF s r"
305 notation
306   OSUF ("suf'_0") and
307   OSUF (infix "suf0" 51) and
308   OSUF (infix " $\supset_\phi$ " 51)
309
310 abbreviation ONSF :: " $\sigma_i \Rightarrow \sigma_i \Rightarrow \beta_i$ " where
311   "ONSF r s ==  $\neg(r \subset_\phi s)$ "
312 notation
313   ONSF ("nsf'_0") and
314   ONSF (infix "nsf0" 51) and
315   ONSF (infix " $\sim\subset_\phi$ " 51)

```

HOL.eq, must be defined for type  $\sigma_i$ , or the logic for type  $\sigma_i$  will be inconsistent. The Axiom of Extension defines HOL.eq for  $\sigma_i$ , but because the formula  $(r \subset_i s \wedge s \subset_i r)$  is used so much to prove set equality, it is convenient to have an equality operator for this formula.

The notation for equal subsets is  $r \subseteq_\epsilon s$ . It is an abbreviation because with `definition`,  $\subseteq_\epsilon$  would be two levels away from the subset formula, and it is the subset formula that is heavily used by the `simp` rules, as has already been mentioned.

$(\mathcal{V}\mathcal{V})$  (NOTATION) 3.1.10. (es0 equal subsets)

```

329 abbreviation OES :: " $\sigma_i \Rightarrow \sigma_i \Rightarrow \beta_i$ " where
330   "OES r s == (r  $\subset_i$  s  $\wedge$  s  $\subset_i$  r)"
331 notation
332   OES ("es0'_0") and
333   OES (infix "es0" 51) and
334   OES (infix " $\subseteq_\epsilon$ " 51)

```

$(\ominus)$  (THEOREM) 3.1.11. (ss0 equals ssf0)

```

338 theorem OSS, ·EQ, ·OSSFN [simp]:
339   "(r  $\subset_i$  s) = (r  $\subset_\phi$  s)"
340   by(metis OSS_def)
341
342 theorem
343   " $\forall r. \forall s. (r \subset_i s) = (r \subset_\phi s)$ "
344   by(simp)

```

If we make the following Theorem a `simp` rule, it will never get used because each side of the conjunction in the theorem will be rewritten using the rule of Theorem 3.1.11. If we put 3.1.12 before 3.1.11 and make it a `simp` rule, it will only get used for theorems that precede 3.1.11. Consequently, we recognize the rewriting on the wall and place Theorem 3.1.11 before 3.1.12, and also put 3.1.11 immediately after the subset operator definition and abbreviation.

Additionally, much of the proving done with the `simp` rules is done with  $\in_i$  at the element level, and many times, for two sets  $r$  and  $s$ , though  $r = s$  cannot automatically be proved directly,  $(r \subset_i s \wedge s \subset_i r)$

can be proved, which requires only one additional step to prove that  $r = s$ . So, we do not want Theorem 3.1.12 as a `simp` rule anyway.

As to trying to get the simplifier to replace  $(r = s)$  with  $(r \subset_l s \wedge s \subset_l r)$ , that cannot be done. It should be obvious why we would not want to even think of doing such a thing, where us not being able to do it is what tells us that it should be obvious that we should not want to do it.

**( $\ominus$ )** (THEOREM) 3.1.12. (ss0 equality)

```

368 | theorem 0ss.eqN:
369 |   "(r ⊂l s ∧ s ⊂l r) = (r = s)"
370 |   by(metis Ax.xN 0ss-def)
371 |
372 | theorem
373 |   "∀r.∀s.(r ⊂l s ∧ s ⊂l r) = (r = s)"
374 |   by(metis Ax.xN 0ss-def)
375 |
376 | theorem "(r ⊂l s ∧ s ⊂l r) = (r = s)"
377 |   apply(simp) oops
378 |   --"Output: (r ⊂φ s ∧ s ⊂φ r) = z"

```

**( $\ominus$ )** (THEOREM) 3.1.13. (ss0 is reflexive)

```

382 | theorem 0ss.is.reflexiveN:
383 |   "r ⊂l r"
384 |   by(metis 0ss-def)
385 |
386 | theorem
387 |   "∀r. r ⊂l r"
388 |   by(metis 0ss-def)

```

**( $\ominus$ )** (THEOREM) 3.1.14. (ss0 is transitive)

```

392 | theorem 0ss.is.transitiveN:
393 |   "p ⊂l r ∧ r ⊂l s → p ⊂l s"
394 |   by(metis 0ss-def)
395 |
396 | theorem
397 |   "∀p.∀r.∀s. p ⊂l r ∧ r ⊂l s → p ⊂l s"
398 |   by(metis 0ss-def)

```

## 3.2 Axiom of Existence (Empty Set Axiom)

### 3.2.1 (Empty Set Constant, Axiom of Existence)

For a particular first-order language, the FOL specification allows constant symbols to be provided [Bil03, 24.“(6)”.3]. However, no constants are required to be provided, and none are provided in a typical formalization of ZFC sets.

Consequently, it would be preferable to state the Axiom of Existence without using a constant, then define a constant  $\emptyset$  having the property that it contains no elements, and then show that  $\emptyset$  is unique.

However, the Isar commands `defs` and `definition`, which are used to define a constant, require that the constant be meta-equivalent to another term. The empty set constant,  $\emptyset$ , will be of atomic type

$\sigma_t$ , but the property that will be used to define  $\emptyset$ ,  $(\forall u.u \notin \emptyset)$ , is of type `bool`. Obviously, we cannot set these two terms as being equivalent. Consequently, I will use the constant  $\emptyset$  in the Axiom of Existence.

Essentially, this means that rather than there being zero constants in MF-ZFC there will be at least one constant in the language.

$(\mathcal{KT})$  (CONSTANT) 3.2.2. (emS empty set: axiomatized by the Axiom of Existence)

```
424 | consts SEm :: " $\sigma_t$ " ("0")
425 |
426 | notation (input)
427 |   SEm ("emS")
```

Reference for the Axiom of Existence: [Gol96, 76]

$(\alpha\xi)$  (AXIOM) 3.2.3. (Axiom of Existence: the empty set contains no elements)

```
433 | axiomatization where
434 |   Ax·emC: " $x \notin \emptyset$ "
435 |
436 | theorem
437 |   " $\forall x.(x \notin \emptyset)$ "
438 |   by(metis Ax·emC)
```

Constant 3.2.2 introduces the existence of  $\emptyset$ , and Axiom 3.2.3 states that  $\emptyset$  contains no elements, but  $(\exists u.\forall x.x \notin u)$  is still stated as a theorem to allow for some possible compartmentalization of concepts if needed. It is labeled as an axiom so that it doesn't have to be renamed if it becomes an axiom and Axiom 3.2.3 is eliminated by means of extensions by definition [Aga07].

$(\alpha\xi)$  (AXIOM) 3.2.4. (Axiom of Existence: no constant form)

```
448 | theorem Ax·emN:
449 |   " $\exists u.\forall x.(x \notin u)$ "
450 |   by(metis Ax·emC)
```

### 3.2.5 (Empty Set Uniqueness, emS Is a Subset of Every Set)

The next theorem shows that if two sets have the empty set property, then they are equal [Gol96, 76.“Ex.4.7”.4]. The subsequent theorem shows that the empty set constant,  $S_{Em}$ , is unique.

$(\Theta)$  (THEOREM) 3.2.6. (Empty set uniqueness)

```
460 | theorem empty.set.uniquenessN:
461 |   " $(\forall x. x \notin r) \wedge (\forall x. x \notin s) \longrightarrow r = s$ "
462 |   by(metis Ax·xN)
463 |
464 | theorem
465 |   " $\forall r.\forall s.(\forall x. x \notin r) \wedge (\forall x. x \notin s) \longrightarrow r = s$ "
466 |   by(metis empty.set.uniquenessN)
```

$(\Theta)$  (THEOREM) 3.2.7. (emS is unique)



```

470 | theorem SEm.is.uniqueC:
471 |   "(∀x. x ∉l r) ↔ r = ∅"
472 |   by(metis Ax.emC Ax.xN)
473 |
474 | theorem
475 |   "∀r. (∀x. x ∉l r) ↔ r = ∅"
476 |   by(metis SEm.is.uniqueC)

```

**(Θ)** (THEOREM) 3.2.8. (emS ss0 r)

```

480 | theorem SEm.OSs.rC [simp]:
481 |   "∅ ⊂φ r = True"
482 |   apply(simp)
483 |   by(metis SEm.is.uniqueC)
484 |
485 | theorem
486 |   "∀r. (∅ ⊂φ r)"
487 |   by(metis SEm.OSs.rC)

```

### 3.3 Axiom of Pairs

#### 3.3.1 (Unordered Pairs, Axiom, Uniqueness, Existence)

The notation for  $S_{pa}$  is  $\langle a, b \rangle$ .

**(κT)** (CONSTANT) 3.3.2. (paS unordered pair: axiomatized by the Axiom of Pairs)

```

497 | consts Spa :: "σl ⇒ σl ⇒ σl"
498 |
499 | notation
500 |   Spa ("paS") and
501 |   Spa ("({(–), (–)}")")

```

The Axiom of Pairs, Theorem ??, is used with Theorems 3.3.9, 4.2.13, and 4.2.14 to help determine whether a set is a member of a finite set.

**(αξ)** (AXIOM) 3.3.3. (Axiom of Pairs: unordered pairs exist)

```

509 | axiomatization where
510 |   Ax.paC [simp]: "x ∈l {r,s} = (x = r ∨ x = s)"
511 |
512 | theorem
513 |   "∀r.∀s. (∀x. x ∈l {r,s} = (x = r ∨ x = s))"
514 |   by(simp)

```

**(αξ)** (AXIOM) 3.3.4. (Axiom of Pairs: no constant form)

```

518 | theorem Ax.paN:
519 |   "∃u. (∀x. x ∈l u ↔ (x = r ∨ x = s))"
520 |   by(metis Ax.paC)
521 |

```

```

522 | theorem
523 |   "∀r.∀s.∃u.(∀x. x ∈l u ↔ (x = r ∨ x = s))"
524 |   by(metis Ax.paC)
525 |
526 | Book reference: \cite[77.'Ex.4.8{}]{seGol}.
527 |
528 | --"THEOREM (Unordered pair uniqueness)"
529 | theorem upair.uniquenessN:
530 |   "(∀x. x ∈l r1 ↔ (x = p ∨ x = q)) ∧
531 |    (∀x. x ∈l r2 ↔ (x = p ∨ x = q)) → r1 = r2"
532 |   by(metis Ax.xN)
533 |
534 | theorem
535 |   "∀p.∀q.∀r1.∀r2.(∀x. x ∈l r1 ↔ (x = p ∨ x = q)) ∧
536 |     (∀x. x ∈l r2 ↔ (x = p ∨ x = q)) → r1 = r2"
537 |   by(metis Ax.xN)

```

### (⊕) (THEOREM) 3.3.5. (paS is unique)

```

541 | theorem Spa.is.uniqueC:
542 |   "(∀x. x ∈l r ↔ (x = p ∨ x = q)) ↔ r = {p,q}"
543 |   by(metis Ax.xN Ax.paC)
544 |
545 | theorem
546 |   "∀p.∀q.∀r.(∀x. x ∈l r ↔ (x = p ∨ x = q)) ↔ r = {p,q}"
547 |   by(metis Spa.is.uniqueC)

```

### 3.3.6 (Singleton Existence)

Reference for singleton existence: [Gol96, 77."Th.4.1"].

The notation for  $S_{si}$  is  $\langle a \rangle$ .

### (Δ) (DEFINITION) 3.3.7. (siS singleton)

```

557 | definition Ssi :: "σl ⇒ σl" where
558 |   "Ssi r = Spa r r"
559 |
560 | notation
561 |   Ssi ("siS") and
562 |   Ssi ("({_})")
563 |
564 | value "{a}"
565 |   -- "{a,a}"

```

### (⊕) (THEOREM) 3.3.8. (Singletons exist)

```

569 | theorem singletons.existC:
570 |   "∃u.(∀x. x ∈l u ↔ x = r)"
571 |   by(metis Ax.paC)
572 |
573 | theorem
574 |   "∀r.∃u.(∀x. x ∈l u ↔ x = r)"
575 |   by(metis Ax.paC)

```

**( $\ominus$ )** (THEOREM) 3.3.9. (siS exists)

```

579 | theorem SSi.existsC [simp]:
580 |   "x ∈l {r} = (x = r)"
581 |   by(metis Ax.paC SSi-def)
582 |
583 | theorem
584 |   "∀r.∀x. x ∈l {r} = (x = r)"
585 |   by(simp)

```

**( $\ominus$ )** (THEOREM) 3.3.10. (Singleton uniqueness)

```

589 | theorem singleton.uniquenessN:
590 |   "(∀x. x ∈l r ↔ x = p) ∧ (∀x. x ∈l s ↔ x = p) → r = s"
591 |   by(metis Ax.xN)
592 |
593 | theorem
594 |   "∀p.∀r.∀s.(∀x. x ∈l r ↔ x = p) ∧ (∀x. x ∈l s ↔ x = p) → r = s"
595 |   by(metis Ax.xN)

```

**( $\ominus$ )** (THEOREM) 3.3.11. (siS is unique)

```

599 | theorem SSi.is.uniqueC:
600 |   "(∀x. x ∈l r ↔ x = s) ↔ r = {s}"
601 |   by(metis
602 |     SSi-def
603 |     SPa.is.uniqueC)
604 |
605 | theorem
606 |   "∀r.∀s.(∀x. x ∈l r ↔ x = s) ↔ r = {s}"
607 |   by(metis SSi.is.uniqueC)

```

**( $\ominus$ )** (THEOREM) 3.3.12. (siS is a pair)

```

611 | theorem SSi.is.a.pairC [simp]:
612 |   "{r,r} = {r}"
613 |   by(metis SSi-def)
614 |
615 | theorem
616 |   "∀r.{r,r} = {r}"
617 |   by(simp)

```

**( $\ominus$ )** (THEOREM) 3.3.13. (emS equals {r} is false)

```

621 | theorem SEm.NEQ_SEmC [simp]:
622 |   "(∅ = {r}) = False"
623 |   by(metis
624 |     SEm.is.uniqueC
625 |     SSi.is.uniqueC)

```

### 3.3.14 (Unordered Pairs Are Unordered, Element Matching)

$(\Theta)$  (THEOREM) 3.3.15. (paS is unordered)

```

631 | theorem Spa.is.unorderedC [simp]:
632 |   "{r,s} = {s,r}"
633 |   by(metis Spa.is.uniqueC)
634 |
635 | theorem
636 |   "∀r.∀s.({r,s} = {s,r})"
637 |   by(simp)

```

Without `siS` being a function, rather than just an abbreviation, and a function which uses the `simp` rule  $\{r, r\} = \{r\}$ , the next corollary as a `simp` rule would not convert  $\{\{a, b\}, \{c\}\}$  to  $\{\{c\}, \{a, b\}\}$ . This is because  $\{\{a, b\}, \{c\}\}$  would actually be  $\{\{a, b\}, \{c, c\}\}$ , which is lexicographically ordered, and no `simp` rule could be put in place to put the singleton first [NPW13, 178].

Additionally, because  $\{r, s\} = \{s, r\}$  is a `simp` rule,  $\{\{r, s\}, \{p\}\}$  in the corollary will be converted to  $\{\{p\}, \{r, s\}\}$ , which is why the corollary is not needed as a `simp` rule.

$(K\omega)$  (COROLLARY) 3.3.16. (paS  $\{r, s\} \{p\} = \text{paS } \{p\} \{r, s\}$ )

```

652 | corollary Spa.Spa.r.s.Ssi.p.EQ.Spa.Ssi.p.Spa.r.sC:
653 |   "{r,s},{p} = {p},{r,s}"
654 |   by(simp)
655 |
656 | corollary
657 |   "∀p.∀r.∀s. {r,s},{p} = {p},{r,s}"
658 |   by(simp)

```

$(\Theta)$  (THEOREM) 3.3.17. (paS element match)

```

662 | theorem Spa.or.element.matchC :
663 |   "{r1,r2} = {s1,s2} ↔ (r1 = s1 ∧ r2 = s2) ∨ (r1 = s2 ∧ r2 = s1)"
664 |   by(metis Ax.paC Spa.is.unorderedC)
665 |
666 | theorem
667 |   "∀r1.∀r2.∀s1.∀s2.({r1,r2} = {s1,s2} ↔ (r1 = s1 ∧ r2 = s2) ∨
668 |                                     (r1 = s2 ∧ r2 = s1))"
669 |   by(metis Spa.or.element.matchC)

```

## 3.4 Ordered Pairs

### 3.4.1 (Ordered Pair Definition, Notation, Existence)

The notation for  $S_{op}$  is  $\langle a, b \rangle$ .

$(\nu\mathcal{V})$  (NOTATION) 3.4.2. (opS ordered pair)

```

679 | abbreviation (input)
680 |   Sop :: "σl ⇒ σl ⇒ σl" where
681 |   "Sop r s == {{r},{r,s}}"

```

```

682 |
683 | notation
684 |   Sop ("opS")
685 |
686 | syntax "_Sop" :: "σl ⇒ σl ⇒ σl" ("((_,_))")
687 |   translations
688 |   "<r,s>" == "CONST Sop r s"

```

Ordered pairs exist for every  $p$  and  $q$ .

**(Θ)** (THEOREM) 3.4.3. (Ordered pairs exist)

```

694 | theorem opairs.existC:
695 |   "∃u. (∀x. x ∈l u ⟷ x = {r} ∨ x = {r,s})"
696 |   by(metis Spa,is,uniqueC)
697 |
698 | theorem
699 |   "∀r.∀s.∃u.(∀x. x ∈l u ⟷ x = {r} ∨ x = {r,s})"
700 |   by(metis opairs.existC)

```

In the following theorem, the condition  $u = \langle r, s \rangle$  explicitly states that the constant  $S_{op}$  has the property of the Axiom of Pairs. This helps Sledgehammer and `metis`, and because Sledgehammer and `metis` help us, we want to help them in return.

**(Θ)** (THEOREM) 3.4.4. (opS exists)

```

709 | theorem Sop.existsC:
710 |   "∃u. (u = <r,s>) ∧ (∀x. x ∈l u ⟷ x = {r} ∨ x = {r,s})"
711 |   by(simp)
712 |
713 | theorem
714 |   "∀r.∀s.∃u. (u = <r,s>) ∧ (∀x. x ∈l u ⟷ x = {r} ∨ x = {r,s})"
715 |   by(simp)

```

### 3.4.5 (Ordered Pair Uniqueness)

**(Θ)** (THEOREM) 3.4.6. (Ordered pair uniqueness)

```

721 | theorem opair.uniquenessC:
722 |   "(∀x. x ∈l r1 ⟷ (x = {p} ∨ x = {p,q})) ∧
723 |     (∀x. x ∈l r2 ⟷ (x = {p} ∨ x = {p,q})) ⟶ r1 = r2"
724 |   by(metis Ax.xN)
725 |
726 | theorem
727 |   "∀p.∀q.∀r1.∀r2.(∀x. x ∈l r1 ⟷ (x = {p} ∨ x = {p,q})) ∧
728 |     (∀x. x ∈l r2 ⟷ (x = {p} ∨ x = {p,q})) ⟶ r1 = r2"
729 |   by(metis Ax.xN)

```

**(Θ)** (THEOREM) 3.4.7. (opS is unique)

```

733 theorem Sop.is.uniqueC:
734   "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )  $\longleftrightarrow r = \langle p,q \rangle$ "
735 proof assume
736   "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )"
737   thus "r =  $\langle p,q \rangle$ "
738 --">By Sop.EXISTSC, there exists a set u =  $\langle p,q \rangle$  which has the same
739   properties as r. Consequently, because r and u will contain the
740   same elements, then by Ax.xN they are equal.'"
741   by(metis
742     Ax.xN Sop.existsC)
743 next assume
744   "r =  $\langle p,q \rangle$ "
745   thus "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )"
746 --">Again, by Sop.EXISTSC, some u =  $\langle p,q \rangle$  exists, and u has the properties
747   of r expressed in the conclusion. By transitivity, r = u, so the
748   conclusion holds.'"
749   by(metis
750     Sop.existsC)
751 qed
752
753 theorem
754   " $\forall p.\forall q.\forall r. (\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})) \longleftrightarrow r = \langle p,q \rangle$ "
755 --":proof fix p show
756 --": "  $\forall q.\forall r. (\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})) \longleftrightarrow r = \langle p,q \rangle$ "
757 --":proof fix q show
758 --": "  $\forall r. (\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})) \longleftrightarrow r = \langle p,q \rangle$ "
759 --":proof fix r show
760 --": " "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )  $\longleftrightarrow r = \langle p,q \rangle$ "
761 proof assume
762   "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )"
763   thus "r =  $\langle p,q \rangle$ "
764   by(metis
765     Ax.xN Sop.existsC)
766 next assume
767   "r =  $\langle p,q \rangle$ "
768   thus "( $\forall x. x \in_l r \longleftrightarrow (x = \{p\} \vee x = \{p,q\})$ )"
769   by(metis
770     Sop.existsC)
771 qed qed qed qed

```

### 3.4.8 (Ordered Pair Element Matching)

**( $\Theta$ )** (THEOREM) 3.4.9. (opS element match)

```

777 theorem Sop.element.matchC:
778   " $\langle r_1, r_2 \rangle = \langle s_1, s_2 \rangle \longleftrightarrow (r_1 = s_1 \wedge r_2 = s_2)$ "
779   by(metis
780     Ssi_def
781     Spa.or.element.matchC)
782
783 theorem
784   " $\forall r_1.\forall r_2.\forall s_1.\forall s_2. (\langle r_1, r_2 \rangle = \langle s_1, s_2 \rangle) \longleftrightarrow (r_1 = s_1 \wedge r_2 = s_2)$ "
785   by(metis Sop.element.matchC)

```

**3.4.10 (Using Ordered Pairs to Recursively Define n-tuples)**

Implement n-tuples according to [Go196, 80]. Page 81, the paragraph after Exercise 4.12, explains that though n-tuples define unique sets, we cannot yet show that a set  $\{x, y, z\}$  with precisely these elements exists.

## 4 Separation, Union, Power Set

### 4.1 Axiom Schema of Separation

#### 4.1.1 (Separation Set Constant, Notation, and Axiom)

The notation for  $S_{Se}$  is

- $\{\bar{q}\}$  and
- $\{x \in \bar{q} \mid P\}$ .

$(\mathcal{KT})$  (CONSTANT) 4.1.2. (seS separation: axiomatized by the Axiom of Separation)

```
806 | consts SSe :: " $\sigma_i \Rightarrow (\sigma_i \Rightarrow \beta_i) \Rightarrow \sigma_i$ "
807 |
808 | notation (input)
809 |   SSe ("seS")
810 |
811 | notation
812 |   SSe ("( $\{\bar{q}\} \mid P$ )")
```

$(\mathcal{VV})$  (NOTATION) 4.1.3. (Set builder notation: all  $x$  in  $q$  such that  $P$ )

```
816 |
817 | translations " $\{x \in_i q \mid P_x\}$ " => " $\{q \mid (\lambda x. P_x)\}$ "
818 |
```

The set  $q$  in  $\{q \mid P\}$  must be a set which exists, and  $P$  should be a function of type  $(\sigma_i \Rightarrow \beta_i)$ , as shown by the type of  $S_{Se}$ . Additionally,  $P$  should be a formula with a free variable. For example, we could have  $(P \equiv (\lambda x. P_x))$ , where  $P_x$  is a FOL formula with a free variable  $x$ . If  $P$  is applied to a set  $x$  using function application syntax,  $P x$ , then if  $P x$  returns true, and if  $x \in_i q$  is also true, then by the Axiom of Separation,  $x \in_i \{q \mid P\}$ .

Because  $P$  can be any function of type  $(\sigma_i \Rightarrow \text{bool})$ , there is the question of whether a recursive trick can be played to get  $\{q \mid P\} \notin \{q \mid P\}$  with the function application  $(P x)$ . The question is a reminder to us that tinkering with the ZFC axioms, and combining those axioms with the HOL axioms, is fraught with risk. Though the two sets of axioms have so far stood the test of time separately, the two together have not stood the test of time, not to mention that changes have to be made to implement the two ZFC axiom schemes.

$(\mathcal{AX})$  (AXIOM) 4.1.4. (Axiom of Separation: separation sets)

```
838 | axiomatization where
839 |   Ax.seC [simp]: " $\{x \in_i \{q \mid P\}\} = \{x \in_i q \wedge P x\}$ "
840 |
841 | theorem
842 |   " $\forall q. \forall P. (\forall x. x \in_i \{q \mid P\} = \{x \in_i q \wedge P x\})$ "
843 |   by(metis Ax.seC)
```

$(\mathcal{AX})$  (AXIOM) 4.1.5. (Axiom of Separation: no constant form)



```

847 | theorem Ax·seN: "∃u. (∀x. x ∈l u ↔ (x ∈l q ∧ P x))"
848 |   by(metis Ax·seC)
849 |
850 | theorem
851 |   "∀q. ∀P. ∃u. (∀x. x ∈l u ↔ (x ∈l q ∧ P x))"
852 |   by(metis Ax·seC)

```

#### 4.1.6 (Separation Set Builder Notation)

As can be seen from Notation 4.1.3, the notation  $\{x \in_l q \mid P_x\}$  is mapped to  $\{q \mid (\lambda x. P_x)\}$ , and so it is equivalent to  $\{q \mid P\}$  with  $P \equiv (\lambda x. P_x)$ . The formula  $P_x$  in  $\{x \in_l q \mid P_x\}$  should be a FOL formula with a free variable  $x$ . The practical difference between  $\{q \mid P\}$  and  $\{x \in_l q \mid P_x\}$  is that in  $\{x \in_l q \mid P_x\}$ , we can specify  $x$  as the free variable to be used in  $P_x$ .

(But even though the notation  $P_x$  is being used, we must still take care to make sure that the variable substituted for  $x$  in  $\{x \in_l q \mid P_x\}$  is also the desired free variable in  $P_x$ . Additionally,  $\Rightarrow$  cannot be used in place of  $\Rightarrow$  in the `translations` command.)

As an example of how to use the notation, and how it relates to the Axiom of Separation, in  $\{x \in_l q \mid P_x\}$ , let  $P_x \equiv x \neq x$ , then

$$\{x \in_l q \mid P_x\} = \{x \in_l q \mid x \neq x\} = \{q \mid (\lambda x. x \neq x)\}. \quad (4.1.1)$$

It can be seen that for the translation

$$\{q \mid (\lambda x. x \neq x)\} \Rightarrow \{q \mid P\}, \quad (4.1.2)$$

we have  $P = (\lambda x. x \neq x)$ . Using the lambda calculus form, by the Axiom of Separation, we have

$$\forall q. (\forall z. z \in_l \{q \mid (\lambda x. x \neq x)\} \longleftrightarrow z \in_l q \wedge (\lambda x. x \neq x) z). \quad (4.1.3)$$

Because of lambda calculus substitution on the right-hand side, we will have  $z \neq z$  be false for every  $z \in_l q$ , hence  $\{x \in_l q \mid x \neq x\} = \emptyset$ .

If there is no free variable  $x$  in  $P_x$ , then unexpected behavior may result, especially if a theorem using  $\{x \in_l q \mid P_x\}$  is still proved to be true.

(TODO: Refer to an example that shows what the next commented out sentence is supposed to show when there is no free variable  $x$  in  $P$ .)

A simple equality that is useful for converting a formula to lambda calculus for use in a separation set is shown in the following theorem.

**( $\Theta$ )** (THEOREM) 4.1.7. (A lambda calculus equivalency useful for separation sets)

```

902 | theorem
903 |   "P = (λz. P z)"
904 |   by(simp)

```

And now, three different notations for separation sets is shown, along with the formula that gives the notation meaning,  $(x \in_l p \wedge P x)$ , and which requires the use of `Ax·se`.

**( $\Theta$ )** (THEOREM) 4.1.8. (seS equivalent notation)

```

912 | theorem
913 |   "(x ∈l q ∧ P x → x ∈l {q|P}) ∧
914 |   (x ∈l {q|P} → x ∈l {q|(λz. P z)}) ∧
915 |   (x ∈l {q|(λz. P z)} → x ∈l {z ∈l q | P z}) ∧

```

```

916 |   (x ∈l {z ∈l q | P z} → x ∈l q ∧ P x)"
917 |   by(metis Ax·seC)
918 | theorem
919 |   "∀q.∀P.∀x.(x ∈l q ∧ P x → x ∈l {q|P}) ∧
920 |     (x ∈l {q|P} → x ∈l {q | (λz. P z)}) ∧
921 |     (x ∈l {q|(λz. P z)} → x ∈l {z ∈l q | P z}) ∧
922 |     (x ∈l {z ∈l q | P z} → x ∈l q ∧ P x)"
923 |   by(metis Ax·seC)

```

#### 4.1.9 (Separation Set Existence and Uniqueness)

**(Θ)** (THEOREM) 4.1.10. (Separation set uniqueness)

```

929 | theorem sep·set·uniquenessN:
930 |   "(∀x. x ∈l r1 ↔ (x ∈l q ∧ P x)) ∧
931 |     (∀x. x ∈l r2 ↔ (x ∈l q ∧ P x)) → r1 = r2"
932 |   by(metis Ax·xN)
933 |
934 | theorem
935 |   "∀q.∀P.∀r1.∀r2.(∀x. x ∈l r1 ↔ (x ∈l q ∧ P x)) ∧
936 |     (∀x. x ∈l r2 ↔ (x ∈l q ∧ P x)) → r1 = r2"
937 |   by(metis Ax·xN)

```

**(Θ)** (THEOREM) 4.1.11. (seS is unique)

```

941 | theorem Sse·is·uniqueC:
942 |   "(∀x. x ∈l r ↔ (x ∈l q ∧ P x)) ↔ r = {q|P}"
943 |   by(metis Ax·seC Ax·xN)
944 |
945 | theorem
946 |   "∀q.∀P.∀r.(∀x. x ∈l r ↔ (x ∈l q ∧ P x)) ↔ r = {q|P}"
947 |   by(metis Ax·xN Ax·seC)

```

#### 4.1.12 (Basic Examples Using Set Builder Notation)

**(ξπ)** (EXAMPLE) 4.1.13. (emS equals all  $x$  not equal to  $x$ )

```

953 | theorem
954 |   "∀r. ∅ = {x ∈l r | x ≠ x}"
955 |   by(metis Ax·emC Sse·is·uniqueC)

```

**(ξπ)** (EXAMPLE) 4.1.14. (The set  $p$  is not in the set not containing  $p$ )

```

959 | theorem
960 |   "∀r. ∀s. s ∉l {x ∈l r | x ≠ s}"
961 |   by(metis (full_types) Ax·seC)

```

**(ξπ)** (EXAMPLE) 4.1.15. (Singletons are membership equal to their seS)

```

965 | theorem
966 |   "x ∈l {r} ↔ x ∈l {z ∈l {r} | z = r}"
967 |   by(metis (full_types)
968 |     Ax·seC
969 |     SSi·existsC)

```

$(\xi\pi)$  (EXAMPLE) 4.1.16. (Singletons equal their seS)

```

973 | theorem
974 |   "∀r. ({r} = {x ∈l {r} | x = r})"
975 |   by(metis (full_types)
976 |     SSi·is·uniqueC
977 |     SSe·is·uniqueC)

```

$(\xi\pi)$  (EXAMPLE) 4.1.17. (The set containing emS equals its seS)

```

981 | theorem
982 |   "{∅} = {x ∈l {∅} | x = ∅}"
983 |   by(metis (full_types)
984 |     SSi·is·uniqueC
985 |     SSe·is·uniqueC)

```

#### 4.1.18 (An Example of No Free Variable in P)

Suppose a mistake is made, and in the property  $P_x$  used in the set builder notation  $\{x \in_l q \mid P_x\}$ , there is no free variable  $x$ . For example, suppose the following equation is used:

$$\{\emptyset\} = \{x \in_l \{\emptyset\} \mid y = \emptyset\}. \quad (4.1.4)$$

Then  $y$  remains a free variable, and the equation is equivalent to

$$\forall y. \{\emptyset\} = \{x \in_l \{\emptyset\} \mid y = \emptyset\}. \quad (4.1.5)$$

This equivalent equation is used to show that the mistake results in a false equation, since simply negating Equation (4.1.4) does not work, as shown here:

$$\neg(\{\emptyset\} = \{x \in_l \{\emptyset\} \mid y = \emptyset\}) \longleftrightarrow \forall y. \neg(\{\emptyset\} = \{x \in_l \{\emptyset\} \mid y = \emptyset\}). \quad (4.1.6)$$

$(\xi\pi)$  (EXAMPLE) 4.1.19. (When no free  $x$  is in  $P_x$  for set builder notation)

```

1007 | theorem
1008 |   "¬(∀r. ({∅} = {x ∈l {∅} | r = ∅}))"
1009 |   by(metis (lifting, full_types)
1010 |     Ax·emC
1011 |     SSi·existsC
1012 |     SSe·is·uniqueC)

```

## 4.2 Axiom of Unions

### 4.2.1 (General Union Set Constant and Axiom)

The start of the discussion by Goldrei on unions: [Gol96, 82].

The notation for  $U_{Ge}$  is  $\langle \mathbf{U} \rangle$

$(\mathcal{KT})$  (CONSTANT) 4.2.2. ( $geU$  general union: axiomatized by the Axiom of Unions)

```

1024 | consts UGe :: "σi ⇒ σi"
1025 |
1026 | notation
1027 |   UGe ("geU") and
1028 |   UGe ("U")

```

Theorems 4.2.13 and 4.2.14 are *simp* rules, and so if the Axiom of Unions is made a *simp* rule, then some formulas become too complicated when rewritten.

$(\alpha\xi)$  (AXIOM) 4.2.3. (Axiom of Unions)

```

1036 | axiomatization where
1037 |   Ax·unC : "x ∈i ⋃ r = (∃u. x ∈i u ∧ u ∈i r)"
1038 |
1039 | theorem
1040 |   "∀r. ∀x. (x ∈i ⋃ r ↔ (∃u. x ∈i u ∧ u ∈i r))"
1041 |   by(metis Ax·unC)

```

$(\alpha\xi)$  (AXIOM) 4.2.4. (Axiom of Unions: no constant form)

```

1045 | theorem Ax·unN:
1046 |   "∃u. ∀x. (x ∈i u ↔ (∃v. x ∈i v ∧ v ∈i r))"
1047 |   by(metis Ax·unC)
1048 |
1049 | theorem
1050 |   "∀r. ∃u. ∀x. (x ∈i u ↔ (∃v. x ∈i v ∧ v ∈i r))"
1051 |   by(metis Ax·unC)

```

### 4.2.5 (Coordinating *simp* Rules for $geU$ )

The most important *simp* rules for  $geU$  are the permutative rewrite rules, [NPW13, 178]. These three rules are associativity, commutativity, and left-commutativity.

As explained in [NPW13], the simplifier recognizes these three rules, and gives them priority by using them first for simplifications. Consequently, it is important to understand the lexicographic order used by these rules so that other *simp* rules “go with the lexicographic flow”.

As demonstrated by the diagram of [NPW13, 179], for union, an ordered rewriting would proceed as follows:

$$\begin{aligned}
 \bigcup \{ \bigcup \{ b, c \}, a \} & (A) \rightarrow \bigcup \{ b, \bigcup \{ c, a \} \} \\
 & (C) \rightarrow \bigcup \{ b, \bigcup \{ a, c \} \} \\
 & (LC) \rightarrow \bigcup \{ a, \bigcup \{ b, c \} \}.
 \end{aligned} \tag{4.2.1}$$

Because the `simp` rule  $\{r, s\} = \{s, r\}$  is in place, and because sets other than  $\emptyset$  are built up from unordered pairs, it is important to know how the simplifier will order the two elements inside an unordered pair. A sure way to discover what `simp` rules are being invoked, and what lexicographic rules are being used, is to look at the output of the following Isar command after it is applied to a simple proof step.

```
using [[simp_trace]] apply(simp)
```

Three examples of the ordering that will occur before any other `simp` rules are invoked are  $\{b, \{a\}\}$ ,  $\{\{c\}, \{a, b\}\}$ , and  $\{\{b\}, \bigcup a\}$ . If you reverse the order of each of these unordered pairs, then use a command such as

```
theorem "{\bigcup a, \{b\}} = z" apply(simp) oops,
```

and then look at the output of `apply(simp)`, you will see that the order of the elements has been reversed.

Because the three permutative rewrite rules have been added as `simp` rules for `geU`, in general, the starting point for additional `simp` rules for `geU` is an equation left-hand side that is lexicographically ordered enough that it will be used. The right-hand side of the `simp` rules should work together to produce the desired lexicographic order, and the order should be achievable given the high priority of the three permutative rewrite rules.

In the process of labeling theorems as `simp` rules, it is possible to add a `simp` rule that is not needed. If a theorem can be proved with `by(simp)`, then there is some possibility that it is not needed as a `simp` rule.

#### 4.2.6 (geU Simplification Rules and fiS)

Because the finite set constant `fiS` will be defined as a union, all `simp` rules for `geU` should work together with the recursive definition of `fiS`. This will allow the equality of unions of finite sets, and the equality of finite sets to many times be easily proved using only `simp` or `auto`.

The basic pattern for a finite set is that a finite set is a union of nested singletons, with, at most, one pair at the end.

The basic idea is that there are two choices. Pairs should be broken up into singletons, with everything moving to the right, or singletons should be combined to make pairs, with everything moving to the left. The `fiS` definition builds a finite set as a union of singletons. This definition is chosen because the simplifier will order a singleton before a pair, and, on the surface, it appears it should take less thought to expand pairs into singletons, for finite sets and unions of finite sets, than combine singletons into pairs, and not end up confused about what it takes to not be in conflict with the three permutative rewrite rules.

The challenge is to expand unions that fit the pattern of a finite set, but not in a way that conflicts with other `simp` rules that simplify and reduce unions.

#### 4.2.7 (Union Uniqueness)

If  $r_1$  and  $r_2$  both have the property of the Axiom of Unions for  $p$ , then  $r = s$ .

$\left(\Theta\right)$  (THEOREM) 4.2.8. (Union uniqueness)

```
1131 | theorem union_uniquenessN:
1132 |   "( $\forall x. x \in_i r_1 \longleftrightarrow (\exists u. x \in_i u \wedge u \in_i p)$ )  $\wedge$ 
1133 |     ( $\forall x. x \in_i r_2 \longleftrightarrow (\exists u. x \in_i u \wedge u \in_i p)$ )  $\longrightarrow r_1 = r_2$ "
1134 |   by(metis Ax.xN)
1135 |
```

```

1136 | theorem
1137 |   "∀p.∀r1.∀r2.(∀x. x ∈i r1 ↔ (∃u. x ∈i u ∧ u ∈i p)) ∧
1138 |     (∀x. x ∈i r2 ↔ (∃u. x ∈i u ∧ u ∈i p)) → r1 = r2"
1139 |   by(metis Ax·xN)

```

If  $r$  has the property of the Axiom of Unions for  $p$ , then  $r = \bigcup p$ .

**(⊖)** (THEOREM) 4.2.9. (geU is unique)

```

1145 | theorem UGe·is·uniqueC:
1146 |   "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s)) ↔ r = ⋃s"
1147 | proof assume
1148 |   "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s))"
1149 |   thus "r = ⋃s"
1150 | --">By Ax·unC, ⋃s has the properties of r in the hypothesis, therefore by
1151 |   Ax·xN, r = ⋃s.'"
1152 |   by(metis
1153 |     Ax·xN
1154 |     Ax·unC)
1155 | next assume
1156 |   "r = ⋃s"
1157 |   thus "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s))"
1158 | --">Let P be the formula stated in the conclusion, then by axiom HOL.SUBST,
1159 |   (λr.P)r = (λr.P)⋃p. Because (λr.P)⋃p is true by Ax·unC, then the
1160 |   conclusion follows.'"
1161 |   by(metis
1162 |     Ax·unC)
1163 | qed
1164 |
1165 | theorem
1166 |   "∀r.∀s.(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s)) ↔ r = ⋃s"
1167 | --": "proof fix r show
1168 | --": "  "∀s.(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s)) ↔ r = ⋃s"
1169 | --": "proof fix s show
1170 | --": "  "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s)) ↔ r = ⋃s"
1171 | proof assume
1172 |   "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s))"
1173 |   thus "r = ⋃s"
1174 |   by(metis
1175 |     Ax·unC Ax·xN)
1176 | next assume
1177 |   "r = ⋃s"
1178 |   thus "(∀x. x ∈i r ↔ (∃u. x ∈i u ∧ u ∈i s))"
1179 |   by(metis
1180 |     Ax·unC)
1181 | qed qed qed

```

#### 4.2.10 (paS Unions Exist, inP Or, geU{r} = r)

The Axiom of Unions only postulates the existence of a union from a set which already exists. One set we have available to us is the unordered pair  $\{r, s\}$ , and because  $S_{Pa}$  and  $U_{Ge}$  are the building blocks for  $U_{Bi}$ ,  $U_{Fi}$ , and  $S_{Fi}$ , which are binary union, finite union, and finite set respectively, then theorems need to be proved about  $U_{Ge}$  unions using  $S_{Pa}$ .

**(⊖)** (THEOREM) 4.2.11. (Unions of paS exist)

```

1193 | theorem unions_of_Spa_existC:
1194 |   "∃u.∀x.(x ∈l u ↔ (∃v. x ∈l v ∧ v ∈l {r,s}))"
1195 |   by(metis Ax.unC)
1196 |
1197 | theorem
1198 |   "∀r.∀s.∃u.∀x.(x ∈l u ↔ (∃v. x ∈l v ∧ v ∈l {r,s}))"
1199 |   by(metis unions_of_Spa_existC)

```

**(⊖)** (THEOREM) 4.2.12. (geU paS exists)

```

1203 | theorem UGe_Spa_existsC:
1204 |   "x ∈l ⋃{r,s} ↔ (∃u. x ∈l u ∧ u ∈l {r,s})"
1205 |   by(metis
1206 |     UGe_is_uniqueC)
1207 |
1208 | theorem
1209 |   "∀r.∀s.∀x. x ∈l ⋃{r,s} ↔ (∃u. x ∈l u ∧ u ∈l {r,s})"
1210 |   by(metis UGe_Spa_existsC)

```

Theorems 4.2.13 and 4.2.14, along with Theorem ?? and 3.3.9, allow simp rules to determine whether a set is a member of a finite set.

**(⊖)** (THEOREM) 4.2.13. (geU siS equals inP)

```

1218 | theorem UGe_Ssi_EQ_PInC [simp]:
1219 |   "x ∈l ⋃{r} = x ∈l r"
1220 |   by(metis
1221 |     Ax.unC
1222 |     Ssi_is_uniqueC)
1223 |
1224 | theorem
1225 |   "∀r.∀x. x ∈l ⋃{r} = x ∈l r"
1226 |   by(metis UGe_Ssi_EQ_PInC)

```

**(⊖)** (THEOREM) 4.2.14. (geU paS equals inP or)

```

1230 | theorem UGe_Spa_EQ_PIn_orC [simp]:
1231 |   "x ∈l ⋃{r,s} = (x ∈l r ∨ x ∈l s)"
1232 |   by(metis
1233 |     Ax.paC
1234 |     UGe_Spa_existsC)
1235 |
1236 | theorem
1237 |   "∀r.∀s.∀x. x ∈l ⋃{r,s} = (x ∈l r ∨ x ∈l s)"
1238 |   by(metis UGe_Spa_EQ_PIn_orC)

```

**(⊖)** (THEOREM) 4.2.15. (geU{r} = r)

```

1242 | theorem UGe_r_EQ_rC [simp]:
1243 |   "⋃{r} = r"
1244 | proof- have
1245 |   "⋃{r} ⊆ε r"

```

```

1246 |   by(simp)
1247 |   thus
1248 |   "U{r} = r"
1249 |   by(metis 0SS,eqN)
1250 | qed
1251 |
1252 | theorem "∀r. U{r} = r"
1253 |   by(simp)

```

#### 4.2.16 (Distribute, geU Permutative Rewrite Rules)

The associative rule of Theorem 4.2.18 will be applied before the `simp` rule of the next theorem, even though 4.2.18 is subsequent to Theorem 4.2.17 [NPW13, 178]. Consequently, 4.2.17 will not get used as a `simp` rule. It is left as a `simp` rule only for the sake of instruction.

The simplifier applied to

$$U\{U\{p, r\}, U\{p, s\}\}, \quad (4.2.2)$$

after 4.2.18 is introduced, will return the value

$$U\{p, U\{r, U\{p, s\}\}\}. \quad (4.2.3)$$

The application of left commute Theorem 4.2.19, and then Theorem 4.2.36, give us what we try to do with one rule, but it is safe to assume that the prover engine knows what is best when it gives priority to the permutative rewrite rules.

**(Θ)** (THEOREM) 4.2.17. (geU p into geU paS)

```

1277 | theorem UGe.p.into.UGe.SPaC [simp]:
1278 |   "U{U{p, r}, U{p, s}} = U{p, U{r, s}}"
1279 | proof- have
1280 |   "U{U{p, r}, U{p, s}} ⊆ε U{p, U{r, s}}"
1281 |   by(simp)
1282 |   thus
1283 |   "U{U{p, r}, U{p, s}} = U{p, U{r, s}}"
1284 |   by(metis
1285 |     0SS,eqN)
1286 | qed
1287 |
1288 | theorem
1289 |   "∀p. ∀r. ∀s. U{U{p, r}, U{p, s}} = U{p, U{r, s}}"
1290 |   by(metis UGe.p.into.UGe.SPaC)

```

**(Θ)** (THEOREM) 4.2.18. (geU is associative)

```

1294 | theorem UGe.is.associativeC [simp]:
1295 |   "U{U{p, r}, s} = U{p, U{r, s}}"
1296 | proof- have
1297 |   "U{U{p, r}, s} ⊆ε U{p, U{r, s}}"
1298 |   by(simp)
1299 |   thus
1300 |   "U{U{p, r}, s} = U{p, U{r, s}}"
1301 |   by(metis

```



```

1302 |   OSs·eqN)
1303 | qed
1304 |
1305 | theorem
1306 |   "∀p.∀r.∀s. U{U{p,r},s} = U{p,U{r,s}}"
1307 |   by(metis UGe·is·associativeC)

```

Commutativity is taken care of by Theorem 3.3.15, so for geU, we only need a rule for left commute.

**(Θ)** (THEOREM) 4.2.19. (geU left commute)

```

1314 | theorem UGe·left·commuteC [simp]:
1315 |   "U{p,U{r,s}} = U{r,U{p,s}}"
1316 |   by(metis
1317 |     SPa·is·unorderedC
1318 |     UGe·is·associativeC)
1319 |
1320 | theorem
1321 |   "∀p.∀r.∀s. U{p,U{r,s}} = U{r,U{p,s}}"
1322 |   by(simp)

```

#### 4.2.20 (Pseudo Associate and Commute)

**(Θ)** (THEOREM) 4.2.21. (geU{siS, paS} left commute)

```

1328 | theorem UGe·SSi-SPa·left·commuteC [simp]:
1329 |   "U{{p},{r,s}} = U{{r},{p,s}}"
1330 | proof-
1331 |   have
1332 |     "U{{p},{r,s}} ⊆ε U{{r},{p,s}}"
1333 |     by(simp)
1334 |   thus
1335 |     "U{{p},{r,s}} = U{{r},{p,s}}"
1336 |     by(metis
1337 |       OSs·eqN)
1338 | qed
1339 |
1340 | theorem
1341 |   "U{{p},{r,s}} = U{{r},{p,s}}"
1342 |   by(simp)

```

**(Θ)** (THEOREM) 4.2.22. (geU{paS, paS} inside commute)

```

1346 | theorem UGe·SPa-SPa·inside·commuteC [simp]:
1347 |   "U{{p,q},{r,s}} = U{{p,r},{q,s}}"
1348 | proof-
1349 |   have
1350 |     "U{{p,q},{r,s}} ⊆ε U{{p,r},{q,s}}"
1351 |     by(simp)
1352 |   thus
1353 |     "U{{p,q},{r,s}} = U{{p,r},{q,s}}"
1354 |     by(metis
1355 |       OSs·eqN)

```

```

1356 | qed
1357 |
1358 | theorem
1359 |   "∀p.∀q.∀r.∀s. ⋃{{p,q},{r,s}} = ⋃{{p,r},{q,s}}"
1360 |   by(simp)

```

If the order of the left-hand side and right-hand side is reversed in Theorem 4.2.23, then it will lead to nontermination due to commutativity, as explained in [NPW13, 178].

Theorem 4.2.23 shows that one theorem can be proved with `by(simp)` and still be needed as a `simp` rule. An example of the use of 4.2.23 would be

$$\bigcup \left\{ \left\{ p, \bigcup q \right\}, \{k, \{l\}\} \right\} \equiv \bigcup \left\{ \{p\}, \left\{ \bigcup q, \{k, \{l\}\} \right\} \right\}. \quad (4.2.4)$$

**( $\ominus$ )** (THEOREM) 4.2.23. (`geU{paS, siS}` associates)

```

1373 | theorem Uge_Spa-Ssi_associatesC [simp]:
1374 |   "⋃{{p,r},{s}} = ⋃{{p},{r,s}}"
1375 |   by(simp)
1376 |
1377 | theorem
1378 |   "∀p.∀r.∀s. ⋃{{p,r},{s}} = ⋃{{p},{r,s}}"
1379 |   by(simp)

```

#### 4.2.24 (Reducing paS to siS, Eliminating emS)

If the duplicate element in the following theorem was changed to be lexicographically least or greatest, then Theorems 3.3.12 and 4.2.22 would isolate the duplicate element and reduce it to a singleton. Therefore, we only need the case for when the duplicate element is neither least nor greatest.

**( $\ominus$ )** (THEOREM) 4.2.25. (`geU{paS, paS}` twin isolate)

```

1390 | theorem Uge_Spa-Spa_twin_isolateC [simp]:
1391 |   "⋃{{p,r},{r,s}} = ⋃{{r},{p,s}}"
1392 |   by(metis
1393 |     Ssi.is.a.pairC
1394 |     Spa.is.unorderedC
1395 |     Uge_Spa-Spa_inside commuteC)
1396 |
1397 | theorem
1398 |   "∀p.∀r.∀s. ⋃{{p,r},{r,s}} = ⋃{{r},{p,s}}"
1399 |   by(simp)

```

**( $\ominus$ )** (THEOREM) 4.2.26. (`geU emS = emS`)

```

1403 | theorem Uge_Sem',EQ,Sem'C [simp]:
1404 |   "⋃∅ = ∅"
1405 |   by(metis
1406 |     Ax.emC
1407 |     Uge.is.uniqueC)

```

$(\Theta)$  (THEOREM) 4.2.27.  $(geU\{emS, r\} = r)$

```

1411 theorem UGe_SEm-r,EQ,·rC [simp]:
1412   "U{0, r} = r"
1413 proof- have
1414   "U{0, r} ⊆ε r"
1415   by(simp)
1416   thus
1417   "U{0, r} = r"
1418   by(metis
1419     0Ss,eqN)
1420 qed
1421
1422 theorem
1423   "∀r. U{0, r} = r"
1424   by(metis UGe_SEm-r,EQ,·rC)

```

#### 4.2.28 (Eliminating All Unions)

$(\Theta)$  (THEOREM) 4.2.29.  $(geU\{siS, siS\} = paS)$

```

1430 theorem UGe_SSi-SSi,EQ,·SPaC [simp]:
1431   "U{{r}, {s}} = {r, s}"
1432   by(metis
1433     UGe_r,EQ,·rC
1434     SSi,is,a,pairC
1435     UGe_SPa-SPainside,commuteC)
1436
1437 theorem
1438   "∀r. ∀s. U{{r}, {s}} = U{{r, s}}"
1439   by(simp)

```

$(\Theta)$  (THEOREM) 4.2.30.  $(geU\{r, \{r, s\}\} = \{r, s\})$

```

1443 theorem UGe_r-r-s,EQ,·r-sC [simp]:
1444   "U{{r}, {r, s}} = {r, s}"
1445   by(metis
1446     SSi,is,a,pairC
1447     SPa,is,unorderedC
1448     UGe_SSi-SSi,EQ,·SPaC
1449     UGe_SSi-SPaleft,commuteC)
1450
1451 theorem
1452   "∀r. ∀s. U{{r}, {r, s}} = {r, s}"
1453   by(simp)

```

#### 4.2.31 (Eliminating a Union of Union, Ordering to the Right)

The following theorem is for when  $r$  is a variable or a constant of type  $\sigma_r$ .

$(\Theta)$  (THEOREM) 4.2.32.  $(geU\ geU\{r, \{s\}\} = geU\{s, geU\ r\})$

```

1461 | theorem UGe·UGe_r_s_EQ·UGe_s-UGe_rc [simp]:
1462 |   "U(U{r,{s}}) = U{s,Ur}"
1463 | proof- have
1464 |   "∀x. x ∈l U(U{r,{s}}) ↔ (∃u. x ∈l u ∧ u ∈l U{r,{s}})"
1465 |   by(smt
1466 |     Ax·unc)
1467 |   hence
1468 |   "U(U{r,{s}}) ⊆ε U{s,Ur}"
1469 |   apply(simp)
1470 |   by(metis
1471 |     Ax·unc)
1472 |   thus
1473 |   "U(U{r,{s}}) = U{s,Ur}"
1474 |   by(metis
1475 |     OSs·eqN)
1476 | qed
1477 |
1478 | theorem
1479 |   "∀r.∀s. U(U{s,{r}}) = U{r,Us}"
1480 |   by(simp)

```

But we also need a corollary for when the singleton comes first in the lexicographic order of the inside union.

$(K\omega)$  (COROLLARY) 4.2.33.  $(\text{geU } \text{geU}\{\{r\}, s\} = \text{geU}\{r, \text{geU } s\})$

```

1487 | corollary UGe·UGe_r_s_EQ·UGe_r-UGe_sc [simp]:
1488 |   "U(U{\{r\},s}) = U{r,Us}"
1489 |   by(simp)
1490 |
1491 | corollary
1492 |   "∀r.∀s. U(U{\{r\},s}) = U{r,Us}"
1493 |   by(simp)

```

A pair is expanded into singletons, since  $\text{fiS}$  is a union of singletons.

$(\Theta)$  (THEOREM) 4.2.34.  $(\text{geU}\{\{p,r\}, s\} = \text{geU}\{\{p\}, \text{geU}\{\{r\}, s\}\})$

```

1499 | theorem UGe_p_r_s_EQ·UGe_p-UGe_r_sc [simp]:
1500 |   "U{\{p,r\},s} = U{\{p\},U{\{r\},s}}"
1501 |   by(simp)
1502 |
1503 | theorem
1504 |   "∀p.∀r.∀s. U{\{p,r\},s} = U{\{p\},U{\{r\},s}}"
1505 |   by(simp)

```

#### 4.2.35 (Eliminating Some Unions)

$(\Theta)$  (THEOREM) 4.2.36.  $(\text{geU}\{r, \text{geU}\{r, s\}\} = \text{geU}\{r, s\})$

```

1511 | theorem UGe_r-UGe_r_s_EQ·UGe_r-sc [simp]:
1512 |   "U{r,U{r,s}} = U{r,s}"
1513 |   by(metis
1514 |     Ax·xN)

```

```

1515 |      UGe·SPa·EQ·PIn·orC)
1516 |
1517 | theorem
1518 |   "U{r,U{r,s}} = U{r,s}"
1519 |   by(simp)

```

**(Θ)** (THEOREM) 4.2.37.  $(geU\{r,geU\{s\}} = geU\{r,s})$

```

1523 | theorem UGe_r-UGe_s-EQ·UGe_r-sC [simp]:
1524 |   "U{r,U{s}} = U{r,s}"
1525 |   by(metis
1526 |     UGe_SSi-SSi-EQ·SPaC
1527 |     UGe·UGe_r-s-EQ·UGe_r-UGe_sC)
1528 |
1529 | theorem
1530 |   "∀r.∀s. U{r,U{s}} = U{r,s}"
1531 |   by(simp)

```

**(KΩ)** (COROLLARY) 4.2.38.  $(geU\{geU\{r\},geU\{s\}} = geU\{r,s})$

```

1535 | corollary UGe-UGe_r-UGe_s-EQ·UGe_r-sC [simp]:
1536 |   "U{U{r},U{s}} = U{r,s}"
1537 |   by(simp)
1538 |
1539 | corollary
1540 |   "∀r.∀s. U{U{r},U{s}} = U{r,s}"
1541 |   by(simp)

```

**(Θ)** (THEOREM) 4.2.39.  $(geU\{geU\{r,s\}} = geU\{r,s})$

```

1545 | theorem UGe-UGe_r-s-EQ·UGe_r-sC [simp]:
1546 |   "U{U{r,s}} = U{r,s}"
1547 |   by(metis
1548 |     SSi·is·a·pairC
1549 |     SPa·is·unorderedC
1550 |     UGe_r-UGe_r-s-EQ·UGe_r-sC)
1551 |
1552 | theorem
1553 |   "∀r.∀s. U{U{r,s}} = U{r,s}"
1554 |   by(simp)

```

#### 4.2.40 (Miscellaneous Results)

**(Θ)** (THEOREM) 4.2.41.  $(p \text{ in } r \text{ implies } p \text{ a subset of } geU\ r)$

```

1560 | theorem p·in·r·IMP·p·a·subset·of·UGe·rC:
1561 |   "p ∈l r → p ⊂l Ur"
1562 |   by(metis Ax·unC 0Ss-def)
1563 |
1564 | theorem
1565 |   "∀p.∀r. p ∈l r → p ⊂l Ur"
1566 |   by(metis Ax·unC 0Ss-def)

```

$(\Theta)$  (THEOREM) 4.2.42. ( $\text{geU}\{x, \{x\}$  exists)

```

1570 | theorem UGe-x-x_existsC:
1571 |   "∃u. u =  $\bigcup\{x, \{x\}\}$ "
1572 |   by(metis Ax·unC)
1573 |
1574 | theorem
1575 |   "∀x. ∃u. u =  $\bigcup\{x, \{x\}\}$ "
1576 |   by(metis Ax·unC)

```

#### 4.2.43 (biU Binary Union)

The binary union  $U_{Bi}$  is introduced as a notational convenience, and the identifier  $U_{Bi}$  will only be used in the name of a theorem when the  $\cup_i$  notation is also being used. To emphasize that  $U_{Ge}$  is the only union operator, and to get good at reasoning with it,  $U_{Ge}$  will be used much of the time to do binary unions.

The notation for  $U_{Bi}$  is  $\langle \text{bold} \rangle \langle \text{union} \rangle$

$(\mathcal{V}\mathcal{V})$  (NOTATION) 4.2.44. (biU Binary Union)

```

1591 | abbreviation (input)
1592 |   UBi :: " $\sigma_i \Rightarrow \sigma_i \Rightarrow \sigma_i$ " where "UBi r s ==  $\bigcup\{r, s\}$ "
1593 |
1594 | notation
1595 |   UBi ("bi'_U") and
1596 |   UBi (infixl "biU" 65) and
1597 |   UBi (infixl " $\cup_i$ " 65)
1598 |
1599 | --"Undefined notation."
1600 | --" $\bigcup\{\}$ "

```

Because  $U_{Bi}$  is merely an abbreviation for  $\bigcup\{p, q\}$ , theorems in this are restatements of what has already been proved.

$(K\omega)$  (COROLLARY) 4.2.45. (biU commutes)

```

1607 | corollary
1608 |   "r  $\cup_i$  s = s  $\cup_i$  r"
1609 |   by(simp)
1610 |
1611 | corollary
1612 |   "∀r. ∀s. r  $\cup_i$  s = s  $\cup_i$  r"
1613 |   by(simp)

```

$(K\omega)$  (COROLLARY) 4.2.46. (biU left commutes)

```

1617 | corollary
1618 |   "p  $\cup_i$  (r  $\cup_i$  s) = r  $\cup_i$  (p  $\cup_i$  s)"
1619 |   by(simp)
1620 |
1621 | corollary
1622 |   "∀p. ∀r. ∀s. p  $\cup_i$  (r  $\cup_i$  s) = r  $\cup_i$  (p  $\cup_i$  s)"
1623 |   by(simp)

```

$(\kappa\omega)$  (COROLLARY) 4.2.47. (biU is associative)

```

1627 corollary
1628   "(p ∪l r) ∪l s = p ∪l (r ∪l s)"
1629   by(simp)
1630
1631 corollary
1632   "∀p.∀r.∀s.(p ∪l r) ∪l s = p ∪l (r ∪l s)"
1633   by(simp)

```

$(\kappa\omega)$  (COROLLARY) 4.2.48. (biU distribute into union)

```

1637 corollary
1638   "p ∪l (r ∪l s) = (p ∪l r) ∪l (p ∪l s)"
1639   by(simp)
1640
1641 corollary
1642   "∀p.∀r.∀s. p ∪l (r ∪l s) = (p ∪l r) ∪l (p ∪l s)"
1643   by(simp)

```

## 4.3 Finite Sets

### 4.3.1 (Finite set fiS, Equality Examples)

The notation for  $S_{\text{Fi}}$  is  $\langle \langle \_ \rangle \rangle$ .

$(\rho\delta)$  (RDEFINITION) 4.3.2. (fiS finite set)

```

1653 fun S_Fi :: "σΛ ⇒ σl" where
1654   "S_Fi [] = ∅"
1655 | "S_Fi (r#rs) = (∪{{r}, S_Fi rs})"
1656
1657 notation (input)
1658   S_Fi ("fiS")
1659
1660 syntax "_S_Fi" :: "σl ⇒ σl ⇒ args ⇒ σl" ("({{_,_,_}})")
1661
1662 translations
1663   "{r,s,ss}" == "CONST S_Fi (r#s#[ss])"

```

$(\xi\pi)$  (EXAMPLE) 4.3.3. (Using geU and paS to build  $\{a, b, c\}$ )

```

1667 theorem "∪{{a,b},{c}} = {a,b,c} ∧
1668         (x ∈l {a,b,c} ↔ x = a ∨ x = b ∨ x = c)"
1669   by(simp)
1670
1671 value "∪{{a,b},{c}}"
1672   --"∪{{a,b},{c,c}}"
1673 value "{a,b,c}"
1674   --"∪{{a,a},∪{{b,b},∪{{c,c},∅}}}"

```

Even though the definition of `fiS` uses `List.list`, with which order and repetition matters, because order and repetition does not matter in an unordered pair, then it does not matter with `fiS`.

The proof method `simp` behaves as if it is of type `calculator`, (`simp::calculator`), and a person who does not know what is happening under the hood of the prover engine might wonder whether anything is actually happening of significance.

With `simp`, the activity becomes, rather than proving, rewriting based on equivalencies that have already been proved, and the last `theorem` in Example 4.3.4 shows, by using `del` with `simp`, the rules which `simp` will use after it is allowed to use them. Without using `simp`, for automatic proving, we would need to use a method such as `metis`, which would need to use at least some of the theorems listed after `del`.

That two sets are equal, even though the order of the elements may be different, and even though there may be duplicate elements, is a simple concept to us, and it would annoy us if we could not make it appear that the prover engine does not require powerful, recursive abilities to deal with, what to us, is simple.

$(\xi\pi)$  (EXAMPLE) 4.3.4. (`fiS` has no order, repetitions do not matter)

```

1699 theorem "{a, z, f, g, m, f, t} = {z, g, m, z, t, a, f}"
1700   by(simp)
1701
1702 theorem "{z, a, {k, {l}}} = {{k, {l}}, z, {k, {l}}, a}"
1703   by(simp)
1704
1705 theorem "{k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {s, t, s, t, t}}}}}}}}}}}"
1706   ∈l
1707   {{a, b, {m}}, {a, b, {m}}, d, z, b, a,
1708    {k, k, {l, {m, m, {n, {a, {a, z, {x, {p, {q, {r, {s, s, s, t, s, t}}}}}}}}}}}"
1709   by(simp)
1710
1711 theorem "{{a, b, {m}}, {a, b, {m}}, d, z, b, a,
1712          {k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {t, s, t}}}}}}}}}}}"
1713   =
1714   {{k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {s, t}}}}}}}}}}}, a, z, b,
1715    {k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {t, s}}}}}}}}}}}, {a, {m}, b}, d}"
1716   by(simp)
1717
1718 theorem "{{a, b, {m}}, {a, b, {m}}, d, z, b, a,
1719          {k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {t, s, t}}}}}}}}}}}"
1720   =
1721   {{k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {s, t}}}}}}}}}}}, a, z, b,
1722    {k, {l, {m, {n, {a, {a, z, {x, {p, {q, {r, {t, s}}}}}}}}}}}, {a, {m}, b}, d}"
1723   apply(simp del:
1724     SSi·is·a·pairC
1725     UGe·SEm·r·EQ·rC
1726     SPa·is·unorderedC
1727     UGe·left·commuteC
1728     UGe·SSi·SSi·EQ·SPaC
1729     UGe·r·r·s·EQ·r·sC
1730     UGe·SPa·SSi·associatesC
1731     UGe·SSi·SPa·left·commuteC)
1732   by(simp)

```

$(\xi\pi)$  (EXAMPLE) 4.3.5. (`geU` unions of `fiS`)



```

1736 theorem
1737   "U{p,U{r,s}} = U{p,r,s}"
1738   by(simp)
1739
1740 theorem
1741   "U{p,U{q,U{r,s}}} = U{p,q,r,s}"
1742   by(simp)
1743
1744 theorem
1745   "{a,b,a,d} ∪i {a,c,m,c} = {m,d,c,b,a}"
1746   by(simp)
1747
1748 theorem
1749   "U{{a,b,c,d,c},{a,m,{n},p}} = {a,b,c,d,m,{n},p}"
1750   by(simp)

```

### 4.3.6 (Converting to Cons)

The definition of `fiS` is based on the operation  $S_{Fi}(r\#rs)$ , so  $[r]@rs$  is converted to  $r\#rs$ .

**( $\Theta$ )** (THEOREM) 4.3.7.  $(fiS([r] @ rs) = fiS(r \# rs))$

```

1759 theorem SFi, 'r', rs, EQ, SFi, r, rsc:
1760   "SFi([r] @ rs) = SFi(r # rs)"
1761   by(simp)
1762
1763 theorem
1764   "∀r.∀rs. SFi([r] @ rs) = SFi(r # rs)"
1765   by(simp)

```

Likewise for  $rs@[r]$ , where the right-hand side of 4.3.8 simplifies to  $\bigcup\{S_{Fi} rs, \{r\}\}$ , which fits the pattern of a `fiS`. The induction method has to be used because `fiS` is defined using `List.list.Cons`, rather than `List.append`.

**( $\Theta$ )** (THEOREM) 4.3.8.  $(fiS(rs @ [r]) = fiS(r \# rs))$

```

1774 theorem SFi, rs, r, EQ, SFi, r, rsc:
1775   "SFi(rs @ [r]) = SFi(r # rs)"
1776   apply(induction rs arbitrary: r)
1777   by(auto)
1778
1779 theorem
1780   "∀r.∀rs. SFi(rs @ [r]) = SFi(r # rs)"
1781   by(metis SFi, rs, r, EQ, SFi, r, rsc)

```

What is proved next does not need to be proved because the definition of `fiS`, along with Theorem 4.2.32, will produce the right-hand side.

**( $\Theta$ )** (THEOREM) 4.3.9.  $(geU(fiS(r \# rs)) = geU\{r, geU(fiS rs)\})$

```

1788 theorem UGe, SFi, r, rs, EQ, UGe, r, UGe, SFi, rsc:
1789   "U(SFi(r # rs)) = U\{r, U(SFi rs)\}"
1790   by(simp)
1791
1792 theorem
1793   "∀r.∀rs. U(SFi(r # rs)) = U\{r, U(SFi rs)\}"
1794   by(simp)

```

The same here, other than, additionally, `List.append` simplification rules are used to convert the `append` to a `Cons`.

$(\ominus)$  (THEOREM) 4.3.10.  $(\text{geU}(\text{fiS}([r] @ \text{rs}))) = \text{geU}\{r, \text{geU}(\text{fiS } \text{rs})\}$

```

1801 | theorem U_Ge_S_Fi_r',rs',EQ,U_Ge_r-U_Ge_S_Fi_rs_^C:
1802 |   "U(S_Fi([r] @ rs)) = U{r,U(S_Fi rs)}"
1803 |   by(simp)
1804 |
1805 | theorem
1806 |   "\r.\Vrs.U(S_Fi([r] @ rs)) = U{r,U(S_Fi rs)}"
1807 |   by(simp)

```

### 4.3.11 (Converting Appends to a Union, Simplifying Unions of `fiS`)

$(\ominus)$  (THEOREM) 4.3.12.  $(\text{geU}\{\text{fiS}\} = \text{fiS})$

```

1813 | theorem U_Ge_S_Fi_ EQ,S_Fi^C [simp]:
1814 |   "U{S_Fi rs} = S_Fi rs"
1815 |   apply(induction rs)
1816 |   by(auto)
1817 |
1818 | theorem
1819 |   "\rs.U{S_Fi rs} = S_Fi rs"
1820 |   by(simp)

```

$(\ominus)$  (THEOREM) 4.3.13.  $(\text{fiS } \text{append } \text{to } \text{pair})$

```

1824 | theorem S_Fi_append_to_pair^C [simp]:
1825 |   "S_Fi(rs @ ss) = U{S_Fi rs,S_Fi ss}"
1826 |   apply(induction rs)
1827 |   by(auto)
1828 |
1829 | theorem
1830 |   "\rs.\Vss. S_Fi(rs @ ss) = U{S_Fi rs,S_Fi ss}"
1831 |   by(simp)

```

$(\ominus)$  (THEOREM) 4.3.14.  $(\text{geU}\{\text{geU } \text{fiS}\} = \text{geU } \text{fiS})$

```

1835 | theorem U_Ge_U_Ge_S_Fi_ EQ,U_Ge_S_Fi^C [simp]:
1836 |   "U{U(S_Fi rs)} = U(S_Fi rs)"
1837 |   apply(induction rs)
1838 |   by(auto)
1839 |
1840 | theorem
1841 |   "\rs.U{U(S_Fi rs)} = U(S_Fi rs)"
1842 |   by(simp)

```

$(\ominus)$  (THEOREM) 4.3.15.  $(\text{geU}\{\text{geU } \text{fiS}, \text{geU } \text{fiS}\} = \text{geU}(\text{geU}\{\text{fiS}, \text{fiS}\}))$

```

1846 theorem UGe.UGe.SFi-UGe.SFi.EQ.·UGe.UGe.SFi-SFic [simp]:
1847   "⋃{⋃(SFi rs), ⋃(SFi ss)} = ⋃(⋃{SFi rs, SFi ss})"
1848   apply(induction rs arbitrary: ss)
1849   apply(auto)
1850   by (metis
1851       UGe.is.associativec
1852       UGe.UGe.r-s-EQ.·UGe.s-UGe.r-c)
1853
1854 theorem
1855   "∀rs.∀ss.⋃{⋃(SFi rs), ⋃(SFi ss)} = ⋃(⋃{SFi rs, SFi ss})"
1856   by(simp)

```

$(\xi\pi)$  (EXAMPLE) 4.3.16. (fiS equations solved using simp only)

```

1860 theorem --"Associativity."
1861   "SFi((ps @ rs) @ ss) = SFi(ps @ (rs @ ss))"
1862   by(simp)
1863
1864 theorem --"Left commute."
1865   "SFi(ps @ (rs @ ss)) = SFi(rs @ (ps @ ss))"
1866   by(simp)
1867
1868 theorem --"Inside commute."
1869   "SFi((ps @ qs) @ (rs @ ss)) = SFi((ps @ rs) @ (qs @ ss))"
1870   by(simp)
1871
1872 theorem --"Order does not matter, and duplicate Cons and appends do not matter."
1873   "SFi((ps @ x # qs @ [f,g,c]) @ (rs @ q # ss) @ (rs @ [d,c,d] @ xs)) =
1874     SFi((((xs @ [q]) @ ps) @ q # qs) @ ([q] @ (x # rs @ [c,d,f,g,c]) @ ss))"
1875   by(simp)

```

## 4.4 Intersection

### 4.4.1 (Definitions, inP And)

If we have a set which exists, and we have a property, then we can claim the existence of a set [Gol96, 84]. For the general intersection of  $r$ , we start with the general union of  $r$ , and we specify a property which only takes elements which are in every set contained in  $r$ .

$(\Delta)$  (DEFINITION) 4.4.2. (geI general intersection)

```

1888 definition IGe :: "σi ⇒ σi" where
1889   "IGe r = {x ∈i ⋃r | ∀p. p ∈i r → x ∈i p}"
1890
1891 notation
1892   IGe ("geI") and
1893   IGe ("∩")

```

For the same reason that binary union is important, binary intersection is important, namely, for the reason that sets are built using singletons and unordered pairs. The notation for infix binary intersection is provided, although most of the preliminary theorems using binary intersection will be stated using the general intersection operator.

$(\forall\forall)$  (NOTATION) 4.4.3. (biI binary intersection)

```

1903 | abbreviation (input)
1904 |   IBi :: "σi ⇒ σi ⇒ σi" where "IBi r s == ⋂{r,s}"
1905 |
1906 | notation
1907 |   IBi ("bi'_I") and
1908 |   IBi (infixl "biI" 70) and
1909 |   IBi (infixl "∩i" 70)

```

**(⊕)** (THEOREM) 4.4.4. (geI siS equals inP)

```

1913 | theorem IGe·SSi·EQ·PInC [simp]:
1914 |   "x ∈i ⋂{r} = (x ∈i r)"
1915 |   apply(unfold IGe_def)
1916 |   by(auto)
1917 |
1918 | theorem
1919 |   "∀r.∀x. x ∈i ⋂{r} = (x ∈i r)"
1920 |   apply(unfold IGe_def)
1921 |   by(auto)

```

**(⊕)** (THEOREM) 4.4.5. (geI paS equals inP and)

```

1925 | theorem IGe·SPa·EQ·PIn·andC [simp]:
1926 |   "x ∈i ⋂{r,s} = (x ∈i r ∧ x ∈i s)"
1927 |   apply(unfold IGe_def)
1928 |   by(auto)
1929 |
1930 | theorem
1931 |   "∀r.∀s.∀x. x ∈i ⋂{r,s} = (x ∈i r ∧ x ∈i s)"
1932 |   apply(unfold IGe_def)
1933 |   by(auto)

```

#### 4.4.6 (NEW: Permutative Rewrite Rules, geI{r} = r)

**(⊕)** (THEOREM) 4.4.7. (NEW: geI is associative)

```

1939 | theorem inter·associate [simp]:
1940 |   "⋂{⋂{p,r},s} = ⋂{p,⋂{r,s}}"
1941 | proof- have
1942 |   "⋂{⋂{p,r},s} ⊆ε ⋂{p,⋂{r,s}}"
1943 |   by(simp)
1944 |   thus
1945 |   "⋂{⋂{p,r},s} = ⋂{p,⋂{r,s}}"
1946 |   by(metis 0Ss,eqN)
1947 | qed

```

**(⊕)** (THEOREM) 4.4.8. (NEW: geI left commute)

```

1951 | theorem inter.left commute [simp]:
1952 |   " $\bigcap\{p, \bigcap\{r, s\}\} = \bigcap\{r, \bigcap\{p, s\}\}$ "
1953 | proof- have
1954 |   " $\bigcap\{p, \bigcap\{r, s\}\} \subseteq_{\epsilon} \bigcap\{r, \bigcap\{p, s\}\}$ "
1955 |   by(simp)
1956 |   thus
1957 |   " $\bigcap\{p, \bigcap\{r, s\}\} = \bigcap\{r, \bigcap\{p, s\}\}$ "
1958 |   by(metis 0SS, eqN)
1959 | qed

```

**( $\Theta$ )** (THEOREM) 4.4.9. ( $\text{geI}\{r\} = r$ )

```

1963 | theorem IGe_r_,EQ, rC [simp]:
1964 |   " $\bigcap\{r\} = r$ "
1965 | proof- have
1966 |   " $\bigcap\{r\} \subseteq_{\epsilon} r$ "
1967 |   by(simp)
1968 |   thus
1969 |   " $\bigcap\{r\} = r$ "
1970 |   by(metis 0SS, eqN)
1971 | qed

```

**( $\Theta$ )** (THEOREM) 4.4.10. (NEW:  $\text{geI}$  of two singletons)

```

1975 | theorem NEWmbyh40a:
1976 |   " $\bigcap\{\{r\}, \{r\}\} = \{r\}$ "
1977 |   by(metis
1978 |     IGe_r_,EQ, rC
1979 |     SSi, is_a_pairC)

```

#### 4.4.11 (NEW: Distribute)

**( $\Theta$ )** (THEOREM) 4.4.12. (NEW: Left distribute)

```

1985 | theorem NEWmbyh57b [simp]:
1986 |   " $\bigcap\{p, \bigcup\{r, s\}\} = \bigcup\{\bigcap\{p, r\}, \bigcap\{p, s\}\}$ "
1987 | proof- have
1988 |   " $\bigcap\{p, \bigcup\{r, s\}\} \subseteq_{\epsilon} \bigcup\{\bigcap\{p, r\}, \bigcap\{p, s\}\}$ "
1989 |   by(simp)
1990 |   thus
1991 |   " $\bigcap\{p, \bigcup\{r, s\}\} = \bigcup\{\bigcap\{p, r\}, \bigcap\{p, s\}\}$ "
1992 |   by(metis 0SS, eqN)
1993 | qed

```

**( $\Theta$ )** (THEOREM) 4.4.13. (NEW: Right distribute)

```

1997 | theorem NEWmbya34a [simp]:
1998 |   " $\bigcap\{\bigcup\{p, r\}, s\} = \bigcup\{\bigcap\{p, s\}, \bigcap\{r, s\}\}$ "
1999 | proof- have
2000 |   " $\bigcap\{\bigcup\{p, r\}, s\} \subseteq_{\epsilon} \bigcup\{\bigcap\{p, s\}, \bigcap\{r, s\}\}$ "

```

```

2001 |   by(simp)
2002 |   thus
2003 |   " $\bigcap\{\bigcup\{p,r\},s\} = \bigcup\{\bigcap\{p,s\},\bigcap\{r,s\}\}$ "
2004 |   by(metis 0SS,eqN)
2005 | qed

```

**(Θ)** (THEOREM) 4.4.14. (NEW: Left distribute into pair)

```

2009 | theorem NEWmbyh59a [simp]:
2010 |   " $\bigcap\{p,\{r,s\}\} = \bigcup\{\bigcap\{p,\{r\}\},\bigcap\{p,\{s\}\}\}$ "
2011 | proof- have
2012 |   " $\bigcap\{p,\{r,s\}\} \subseteq_{\epsilon} \bigcup\{\bigcap\{p,\{r\}\},\bigcap\{p,\{s\}\}\}$ "
2013 |   by(auto)
2014 |   thus
2015 |   " $\bigcap\{p,\{r,s\}\} = \bigcup\{\bigcap\{p,\{r\}\},\bigcap\{p,\{s\}\}\}$ "
2016 |   by(metis 0SS,eqN)
2017 | qed

```

**(Θ)** (THEOREM) 4.4.15. (NEW: Right distribute into pair)

```

2021 | theorem NEWmbyh60a [simp]:
2022 |   " $\bigcap\{\{r,s\},p\} = \bigcup\{\bigcap\{p,\{r\}\},\bigcap\{p,\{s\}\}\}$ "
2023 |   by(simp)

```

#### 4.4.16 (NEW: Example)

**(ξπ)** (EXAMPLE) 4.4.17. (NEW: example)

```

2029 |
2030 | theorem stuffer2 [simp]:
2031 |   "(r =  $\bigcup\{\{r\},s\}) = \text{False}$ "
2032 |   sorry
2033 |   thm "stuffer2"
2034 |
2035 | theorem stuffer3 [simp]:
2036 |   "(r = {s,\{r\}}) = \text{False}"
2037 |   sorry
2038 |   thm "stuffer3"
2039 |
2040 | --"
2041 | a =  $\emptyset$ 
2042 | b =  $\{\emptyset\}$ 
2043 | c =  $\{\{\emptyset\}\}$  or  $\{\emptyset,\{\emptyset\}\}$ 
2044 | d =  $\{\{\{\emptyset\}\}\}$  or  $\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}$ 
2045 | "
2046 |
2047 | theorem --"equal subsets operator"
2048 |   " $\bigcap\{\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\},\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}\},\{\{\emptyset\},\{\emptyset,\{\emptyset\}\},\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}\}$ "
2049 |      $\subseteq_{\epsilon} \{\{\emptyset\},\{\emptyset,\{\emptyset\}\},\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}\}$ "
2050 |   by(simp)
2051 |
2052 | theorem --"equal subsets operator"

```

```

2053 | "⋂{{ 0,{0},{{{0}}} },{ {0},{{0}},{{{0}}} }} ⊆ε {{0},{{{0}}}"
2054 | by(simp)
2055 |
2056 | theorem --"equal"
2057 | "⋂{{ 0,{0},{{{0}}} },{ {0},{{0}},{{{0}}} }} = {{0},{{{0}}}"
2058 | apply(simp)
2059 | oops
2060 |
2061 | theorem "⋂{{a,b,y,z},{m,n,y,z}} = {z,y}"
2062 | apply simp
2063 | oops
2064 |
2065 | theorem "y ∈i ⋂{{a,b,y,z},{m,n,y,z}}"
2066 | by simp
2067 |
2068 | theorem "z ∈i ⋂{{a,b,y,z},{m,n,y,z}}"
2069 | by simp

```

## 4.5 Axiom of Power Sets

### 4.5.1 (Power Set Constant and Axiom)

The start of the discussion by Goldrei on power sets: [Gol96, 82].

The notation for  $\mathcal{P}_S$  is  $\langle P \rangle \langle \hat{\ } \text{isub} \rangle \langle S \rangle$ .

$(\kappa\mathcal{T})$  (CONSTANT) 4.5.2. (pwS power set: axiomatized by the Axiom of Power Sets)

```

2081 | consts  $\mathcal{P}_S$  :: "sT  $\Rightarrow$  sT"
2082 |
2083 | notation (input)
2084 |  $\mathcal{P}_S$  ("pwS")

```

$(\alpha\xi)$  (AXIOM) 4.5.3. (Axiom of Power Sets)

```

2088 | axiomatization where
2089 | Ax.pwc: "(x ∈i  $\mathcal{P}_S(r)$   $\longleftrightarrow$  x  $\subset_i$  r)"
2090 |
2091 | theorem
2092 | "∀r.∀x.(x ∈i  $\mathcal{P}_S(r)$   $\longleftrightarrow$  x  $\subset_i$  r)"
2093 | by(metis Ax.pwc)

```

$(\alpha\xi)$  (AXIOM) 4.5.4. (Axiom of Power Sets: no constant form)

```

2097 | theorem Ax.pwn:
2098 | "∃u.∀x.(x ∈i u  $\longleftrightarrow$  x  $\subset_i$  r)"
2099 | by(metis Ax.pwc)
2100 |
2101 | theorem
2102 | "∀r.∃u.∀x.(x ∈i u  $\longleftrightarrow$  x  $\subset_i$  r)"
2103 | by(metis Ax.pwc)

```

## 4.5.5 (Power Set Uniqueness)

$(\ominus)$  (THEOREM) 4.5.6. (Power set uniqueness)

```

2109 theorem power.set.uniquenessN:
2110   "( $\forall x. x \in_l r_1 \longleftrightarrow x \subset_l p$ )  $\wedge$ 
2111     ( $\forall x. x \in_l r_2 \longleftrightarrow x \subset_l p$ )  $\longrightarrow r_1 = r_2$ "
2112   by(metis Ax.xN)
2113
2114 theorem
2115   " $\forall p. \forall r_1. \forall r_2. (\forall x. x \in_l r_1 \longleftrightarrow x \subset_l p) \wedge$ 
2116     ( $\forall x. x \in_l r_2 \longleftrightarrow x \subset_l p$ )  $\longrightarrow r_1 = r_2$ "
2117   by(metis Ax.xN)

```

$(\ominus)$  (THEOREM) 4.5.7. (pwS is unique)

```

2121 theorem  $\mathcal{P}_S$ .is.uniqueC:
2122   "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )  $\longleftrightarrow r = \mathcal{P}_S(s)$ "
2123 proof assume
2124   "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )"
2125   thus "r =  $\mathcal{P}_S(s)$ "
2126   by(metis
2127     Ax.pwC Ax.xN)
2128 next assume
2129   "r =  $\mathcal{P}_S(s)$ "
2130   thus "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )"
2131   by(metis
2132     Ax.pwC)
2133 qed
2134
2135 theorem
2136   " $\forall s. \forall r. (\forall x. x \in_l r \longleftrightarrow x \subset_l s) \longleftrightarrow r = \mathcal{P}_S(s)$ "
2137 proof fix s show
2138   " $\forall r. (\forall x. x \in_l r \longleftrightarrow x \subset_l s) \longleftrightarrow r = \mathcal{P}_S(s)$ "
2139 proof fix r show
2140   "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )  $\longleftrightarrow r = \mathcal{P}_S(s)$ "
2141 proof assume
2142   "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )"
2143   thus "r =  $\mathcal{P}_S(s)$ "
2144   by(metis
2145     Ax.pwC Ax.xN)
2146 next assume
2147   "r =  $\mathcal{P}_S(s)$ "
2148   thus "( $\forall x. x \in_l r \longleftrightarrow x \subset_l s$ )"
2149   by(metis
2150     Ax.pwC)
2151 qed qed qed

```



**5 \*\*\*\*\* WORKING HERE START \*\*\*\*\***

**5.1 \*\*\*\*\* WORKING START \*\*\*\*\***

**5.2 \*\*\*\*\* WORKING END \*\*\*\*\***

**6 \*\*\*\*\* WORKING HERE END \*\*\*\*\***

$(\iota\sigma)$  (ISAR) 6.0.1. (Theory end)

2162 | end

## Bibliography

- [Aga07] Neithan Agarwaen. Extensions by definitions. Wikipedia, 2007. [http://en.wikipedia.org/wiki/Extension\\_by\\_definitions](http://en.wikipedia.org/wiki/Extension_by_definitions).
- [Bil03] Stefan Bilaniuk. *A Problem Course in Mathematical Logic*. Version 1.6, 2003. <http://euclid.trentu.ca/math/sb/pcml>.
- [Gol96] Derek Goldrei. *Classic Set Theory*. Chapman and Hall/CRC, New York, 1996. <http://www.mcs.open.ac.uk/People/d.c.goldrei>.
- [Nip13] Tobias Nipkow. *Programming and Proving in Isabelle/HOL*. University of Cambridge and Technische Universität München, 2013 edition, February 2013. <http://isabelle.in.tum.de/website-Isabelle2013/>.
- [NPW13] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Berlin, 2013. <http://isabelle.in.tum.de/website-Isabelle2013/>.

## Index

### Symbols

$O_P$

$\text{biU}, U_{\text{Bi}}, \cup_i$   
 $\langle \text{^bold} \rangle \langle \text{union} \rangle \dots \dots \dots 31$

$\text{geU}, U_{\text{Ge}}, \bigcup$   
 $\langle \text{^bold} \rangle \langle \text{Union} \rangle \dots \dots \dots 21$

$\text{inP}, P_{\text{In}}, x \in_i y$   
 $x \langle \text{in} \rangle \langle \text{^isub} \rangle \langle \text{iota} \rangle y \dots \dots \dots 5$

$S_T$

$\text{fiS}, S_{\text{Fi}}, \{a, b, c\}$   
 $\langle \text{lbrace} \rangle \dots \langle \text{rbrace} \rangle \dots \dots \dots 32$

$\text{opS}, S_{\text{Op}}, \langle a, b \rangle$   
 $\langle \text{langle} \rangle a, b \langle \text{rangle} \dots \dots \dots 13$

$\text{paS}, S_{\text{Pa}}, \{a, b\}$   
 $\langle \text{lbrace} \rangle a, b \langle \text{rbrace} \rangle \dots \dots \dots 10$

$\text{pwS}, \mathcal{P}_S$   
 $\langle P \rangle \langle \text{^isub} \rangle \langle S \rangle \dots \dots \dots 40$

$\text{seS}, S_{\text{Se}}, \{q \mid P\}$   
 $\langle \text{lbrace} \rangle q \langle \text{bar} \rangle P \langle \text{rbrace} \rangle \dots \dots \dots 17$

$\text{seS}, S_{\text{Se}}, \{x \in_i q \mid Q\}$   
 $\langle \text{lbrace} \rangle x \langle \text{in} \rangle \langle \text{^isub} \rangle \langle \text{iota} \rangle \langle \text{bar} \rangle P \langle \text{rbrace} \rangle \dots \dots \dots 17$

$\text{siS}, S_{\text{Si}}, \{a\}$   
 $\langle \text{lbrace} \rangle a \langle \text{rbrace} \rangle \dots \dots \dots 11$

$\mathcal{R}_W$

$\text{emS ss0 r} \dots \dots \dots 10$

$\text{emS equals } \{r\} \text{ is false} \dots \dots \dots 12$

$\text{fiS append to pair} \dots \dots \dots 35$

$\text{geI}\{r\} = r \dots \dots \dots 38$

$\text{geU emS} = \text{emS} \dots \dots \dots 27$

$\text{geU geU}\{r, s\} = \text{geU}\{r, \text{geU } s\} \dots \dots \dots 29$

$\text{geU geU}\{r, \{s\}\} = \text{geU}\{s, \text{geU } r\} \dots \dots \dots 28$

$\text{geU paS equals inP or} \dots \dots \dots 24$

$\text{geU siS equals inP} \dots \dots \dots 24$

$\text{geU}\{p, r, s\} = \text{geU}\{p, \text{geU}\{r, s\}\} \dots \dots \dots 29$

$\text{geU}\{r, \{r, s\}\} = \{r, s\} \dots \dots \dots 28$

$\text{geU}\{\text{emS}, r\} = r \dots \dots \dots 28$

$\text{geU}\{\text{fiS}\} = \text{fiS} \dots \dots \dots 35$

$\text{geU}\{\text{geU fiS}, \text{geU fiS}\} = \text{geU}\{\text{geU}\{\text{fiS}, \text{fiS}\}\} \dots \dots \dots 35$

$\text{geU}\{\text{geU fiS}\} = \text{geU fiS} \dots \dots \dots 35$

$\text{geU}\{\text{geU}\{r, s\}\} = \text{geU}\{r, s\} \dots \dots \dots 30$

$\text{geU}\{\text{geU}\{r\}, \text{geU}\{s\}\} = \text{geU}\{r, s\} \dots \dots \dots 30$

$\text{geU}\{\text{paS}, \text{paS}\} \text{ twin isolate} \dots \dots \dots 27$

$\text{geU}\{\text{paS}, \text{siS}\} \text{ associates} \dots \dots \dots 27$

$\text{geU}\{r, \text{geU}\{r, s\}\} = \text{geU}\{r, s\} \dots \dots \dots 29$

$\text{geU}\{r, \text{geU}\{s\}\} = \text{geU}\{r, s\} \dots \dots \dots 30$

$\text{geU}\{r\} = r \dots \dots \dots 24$

$\text{geU}\{\text{siS}, \text{paS}\} \text{ left commute} \dots \dots \dots 26$

$\text{geU}\{\text{siS}, \text{siS}\} = \text{paS} \dots \dots \dots 28$

$\text{geU}$  is associative  $\dots \dots \dots 25$

$\text{geU}$  left commute  $\dots \dots \dots 26$

$\text{geU } p \text{ into geU } \text{paS} \dots \dots \dots 25$

$\text{paS}$  is unordered  $\dots \dots \dots 13$

$\text{siS}$  exists  $\dots \dots \dots 12$

$\text{siS}$  is a pair  $\dots \dots \dots 12$

$\text{ss0 equals ssf0} \dots \dots \dots 7$

---

Axiom of Pairs: unordered pairs exist .....	10
Axiom of Separation: separation sets .....	17