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

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

Contents

1 Preliminary Prerequisite Knowledge, TODOs

1.1 Theory Begin

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

```
theory sTs
imports Complex_Main "i"
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: $\mathtt{P}_{\mathtt{In}}$.
- 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 $\mathtt{S}_{\mathtt{Em}}$, is a constant that represents a set.
- A capital “P’’ in a name, such as $\mathtt{P}_{\mathtt{In}}$, is a function that is a predicate.
- A capital “O’’ in a name, such as $\mathtt{O}_{\mathtt{Ss}}$, 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 $\mathtt{P}_{\mathtt{In}}$, 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 $\mathtt{U}_{\mathtt{Fi}}$,
- **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 `(λx.  P x)` with `\texttt`, (λx. 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 $\forall P.\forall x.(\lambda\text{x. P x})$, 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 $\forall P.\forall x.(\lambda\text{x. P x})$, 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 $\mathtt{U_{Fi}}$, 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 $y \longleftarrow x$.

$\big(\nu\nu\big)$ $\big($Notation$\big)$ 2.2.1. (`impliedby` operator: long left arrow)

```
abbreviation (input)
  "impliedby x y == y ⟶ x"

notation
  impliedby  (infixl "<--" 25) and
  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, σ_ι has been defined for as non-ASCII notation. The subscripted character used in the σ_ι is the Greek letter iota.

$\left(\tau\iota\right)$ (Type) 3.1.2. (sT primitive set type: everything is a set)

```
typedecl sT ("σι")
```

$\left(\tau\iota\right)$ (Type) 3.1.3. (sTL::(sT list), bT::bool)

```
type_synonym sTL = "sT list" ("σΛ")

type_synonym bT = bool ("βι")
```

ZFC sets is specified to have one predicate, which is membership. The ASCII and non-ASCII infix notation for membership are $\mathrm{P_{In}}$ and $\in_\iota$, along with $\mathrm{P_{In}}$f for ASCII function notation. For negation of membership, there is niOp, $\notin_\iota$, and niOpi.

The notation for $\mathrm{P_{In}}$ is `x\<in>\<^isub>\<iota>y`

$\left(\pi\delta\right)$ (Predicate) 3.1.4. (inP membership: axiomatized by subsequent axioms)

```
consts PIn :: "σι ⇒ σι ⇒ βι"

notation
  PIn ("in'_P") and
  PIn (infix "inP" 51) and
  PIn (infix "∈ι" 51)

abbreviation PNi :: "σι ⇒ σι ⇒ βι" where
  "PNi p q == ¬(p ∈ι q)"

notation
  PNi ("ni'_P") and
  PNi (infix "niP" 51) and
  PNi (infix "∉ι"  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

$$(x \in_\iota r \longleftrightarrow x \in_\iota s) \longleftrightarrow (r = s). \tag{3.1.1}$$

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)

```
axiomatization where
  Ax›xᴺ: "(∀x. x ∈ᵢ r ⟷ x ∈ᵢ s) = (r = s)"

theorem
  "∀r.∀s.(∀x. x ∈ᵢ r ⟷ x ∈ᵢ s) = (r = s)"
  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 $\in_\iota$, 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 $\subset_\iota$ 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.

(Δ) (Definition) 3.1.8. (`ss0` subset, `su0` superset, `ns0` not a subset)

```
definition 0ₛₛ :: "σᵢ ⇒ σᵢ ⇒ βᵢ" where
  "0ₛₛ r s = (∀x. x ∈ᵢ r ⟶ x ∈ᵢ s)"
notation
  0ₛₛ  ("ss'_0") and
  0ₛₛ  (infix "ss0" 51) and
  0ₛₛ  (infix "⊂ᵢ" 51)

abbreviation (input) 0ₛᵤ :: "σᵢ ⇒ σᵢ ⇒ βᵢ" where
  "0ₛᵤ r s == 0ₛₛ s r"
notation
  0ₛᵤ  ("su'_0") and
  0ₛᵤ  (infix "su0" 51) and
  0ₛᵤ  (infix "⊃ᵢ" 51)

abbreviation 0ₙₛ :: "σᵢ ⇒ σᵢ ⇒ βᵢ" where
  "0ₙₛ r s == ¬(r ⊂ᵢ s)"
notation
  0ₙₛ  ("ns'_0") and
  0ₙₛ  (infix "ns0" 51) and
  0ₙₛ  (infix "~⊂ᵢ" 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)

```
abbreviation OSsf :: "σι ⇒ σι ⇒ βι" where
  "OSsf r s == (∀x. x ∈ι r ⟶ x ∈ι s)"
notation
  OSsf  ("ssf'_0") and
  OSsf  (infix "ssf0" 51) and
  OSsf  (infix "⊂ϕ" 51)

abbreviation (input) OSuf :: "σι ⇒ σι ⇒ βι" where
  "OSuf r s == OSsf s r"
notation
  OSuf  ("suf'_0") and
  OSuf  (infix "suf0" 51) and
  OSuf  (infix "⊃ϕ" 51)

abbreviation ONsf :: "σι ⇒ σι ⇒ βι" where
  "ONsf r s == ¬(r ⊂ϕ s)"
notation
  ONsf  ("nsf'_0") and
  ONsf  (infix "nsf0" 51) and
  ONsf  (infix "~⊂ϕ" 51)
```

HOL.eq, must be defined for type σ_ι, or the logic for type σ_ι will be inconsistent. The Axiom of Extension defines HOL.eq for σ_ι, but because the formula $(r \subset_\iota s \wedge s \subset_\iota r)$ 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.

$(\nu\nu)$ (Notation) 3.1.10. (es0 equal subsets)

```
abbreviation OEs :: "σι ⇒ σι ⇒ βι" where
  "OEs r s == (r ⊂ι s ∧ s ⊂ι r)"
notation
  OEs  ("es0'_0") and
  OEs  (infix "es0" 51) and
  OEs  (infix "⊆ϵ" 51)
```

(Θ) (Theorem) 3.1.11. (ss0 equals ssf0)

```
theorem OSs''EQ''OSsf^N [simp]:
  "(r ⊂ι s) = (r ⊂ϕ s)"
  by(metis OSs_def)

theorem
  "∀r.∀s.(r ⊂ι s) = (r ⊂ϕ s)"
  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 $\in_\iota$ at the element level, and many times, for two sets r and s, though $r = s$ cannot automatically be proved directly, $(r \subset_\iota s \wedge s \subset_\iota r)$

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 $(r \subset_\iota s \wedge s \subset_\iota r)$, 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.

(Θ) (Theorem) 3.1.12. (ss0 equality)

```
theorem 0ss'eqN:
  "(r ⊂ι s ∧ s ⊂ι r) = (r = s)"
  by(metis Ax'xN 0ss_def)

theorem
  "∀r.∀s.(r ⊂ι s ∧ s ⊂ι r) = (r = s)"
  by(metis Ax'xN 0ss_def)

theorem "(r ⊂ι s ∧ s ⊂ι r) = (r = s)"
  apply(simp) oops
  --"Output: (r ⊂∅ s ∧ s ⊂∅ r) = z"
```

(Θ) (Theorem) 3.1.13. (ss0 is reflexive)

```
theorem 0ss'is'reflexiveN:
  "r ⊂ι r"
  by(metis 0ss_def)

theorem
  "∀r. r ⊂ι r"
  by(metis 0ss_def)
```

(Θ) (Theorem) 3.1.14. (ss0 is transitive)

```
theorem 0ss'is'transitiveN:
  "p ⊂ι r ∧ r ⊂ι s ⟶ p ⊂ι s"
  by(metis 0ss_def)

theorem
  "∀p.∀r.∀s. p ⊂ι r ∧ r ⊂ι s ⟶ p ⊂ι s"
  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 ∅ having the property that it contains no elements, and then show that ∅ 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, ∅, will be of atomic type

σ_ι, but the property that will be used to define $\emptyset$, $(\forall u.u \notin_\iota \emptyset)$, 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.

$\left(\kappa\tau\right)$ (Constant) 3.2.2. (emS empty set: axiomatized by the Axiom of Existence)

```
consts S_Em :: "σ_ι" ("∅")

notation (input)
  S_Em  ("emS")
```

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

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

```
axiomatization where
  Ax·em^C: "(x ∉_ι ∅)"

theorem
  "∀x.(x ∉_ι ∅)"
  by(metis Ax·em^C)
```

Constant 3.2.2 introduces the existence of $\emptyset$, and Axiom 3.2.3 states that $\emptyset$ contains no elements, but $(\exists u.\forall x.x \notin_\iota u)$ 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].

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

```
theorem Ax·em^N:
  "∃u.∀x.(x ∉_ι u)"
  by(metis Ax·em^C)
```

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.

$\left(\Theta\right)$ (Theorem) 3.2.6. (Empty set uniqueness)

```
theorem empty·set·uniqueness^N:
  "(∀x. x ∉_ι r) ∧ (∀x. x ∉_ι s) ⟶ r = s"
  by(metis Ax·x^N)

theorem
  "∀r.∀s.(∀x. x ∉_ι r) ∧ (∀x. x ∉_ι s) ⟶ r = s"
  by(metis empty·set·uniqueness^N)
```

$\left(\Theta\right)$ (Theorem) 3.2.7. (emS is unique)

```
theorem S_Em'is'unique^C:
  "(∀x. x ∉_ι r) ⟷ r = ∅"
  by(metis Ax'em^C Ax'x^N)

theorem
  "∀r.(∀x. x ∉_ι r) ⟷ r = ∅"
  by(metis S_Em'is'unique^C)
```

(Θ) (Theorem) 3.2.8. (emS ss0 r)

```
theorem S_Em'0_Ss'r^C [simp]:
  "∅ ⊂_ϕ r = True"
  apply(simp)
  by(metis S_Em'is'unique^C)

theorem
  "∀r.(∅ ⊂_ϕ r)"
  by(metis S_Em'0_Ss'r^C)
```

3.3 Axiom of Pairs

3.3.1 (Unordered Pairs, Axiom, Uniqueness, Existence)

The notation for S_{Pa} is \<lbrace>a,b\<rbrace>.

$(\kappa\tau)$ (Constant) 3.3.2. (paS unordered pair: axiomatized by the Axiom of Pairs)

```
consts S_Pa :: "σ_ι ⇒ σ_ι ⇒ σ_ι"

notation
  S_Pa ("paS") and
  S_Pa ("({(_),(_)})")
```

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.

$(\alpha\xi)$ (Axiom) 3.3.3. (Axiom of Pairs: unordered pairs exist)

```
axiomatization  where
  Ax'pa^C [simp]: "x ∈_ι {r,s} = (x = r ∨ x = s)"

theorem
  "∀r.∀s.(∀x. x ∈_ι {r,s} = (x = r ∨ x = s))"
  by(simp)
```

$(\alpha\xi)$ (Axiom) 3.3.4. (Axiom of Pairs: no constant form)

```
theorem Ax'pa^N:
  "∃u.(∀x. x ∈_ι u ⟷ (x = r ∨ x = s))"
  by(metis Ax'pa^C)
```

```
theorem
  "∀r.∀s.∃u.(∀x. x ∈ι u ⟷ (x = r ∨ x = s))"
  by(metis Ax·pa^C)

Book reference: \cite[77.‘Ex.4.8{’}]{seGol}.

--"THEOREM (Unordered pair uniqueness)"
theorem upair·uniqueness^N:
  "(∀x. x ∈ι r1 ⟷ (x = p ∨ x = q)) ∧
  (∀x. x ∈ι r2 ⟷ (x = p ∨ x = q)) ⟶ r1 = r2"
  by(metis Ax·x^N)

theorem
  "∀p.∀q.∀r1.∀r2.(∀x. x ∈ι r1 ⟷ (x = p ∨ x = q)) ∧
                 (∀x. x ∈ι r2 ⟷ (x = p ∨ x = q)) ⟶ r1 = r2"
  by(metis Ax·x^N)
```

(Θ) (Theorem) 3.3.5. (paS is unique)

```
theorem S_Pa·is·unique^C:
  "(∀x. x ∈ι r ⟷ (x = p ∨ x = q)) ⟷ r = {p,q}"
  by(metis Ax·x^N Ax·pa^C)

theorem
  "∀p.∀q.∀r.(∀x. x ∈ι r ⟷ (x = p ∨ x = q)) ⟷ r = {p,q}"
  by(metis S_Pa·is·unique^C)
```

3.3.6 (Singleton Existence)

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

The notation for S_{Si} is \<lbrace>a\<rbrace>.

(Δ) (Definition) 3.3.7. (siS singleton)

```
definition S_Si :: "σι ⇒ σι" where
  "S_Si r = S_Pa r r"

notation
  S_Si  ("siS") and
  S_Si  ("({(_)})")

value "{a}"
       --"{a,a}"
```

(Θ) (Theorem) 3.3.8. (Singletons exist)

```
theorem singletons·exist^C:
  "∃u.(∀x. x ∈ι u ⟷ x = r)"
  by(metis Ax·pa^C)

theorem
  "∀r.∃u.(∀x. x ∈ι u ⟷ x = r)"
  by(metis Ax·pa^C)
```

(Θ) (THEOREM) 3.3.9. (siS exists)

```
theorem S_Si‹exists^C [simp]:
  "x ∈_ι {r} = (x = r)"
  by(metis Ax‹pa^C S_Si_def)

theorem
  "∀r.∀x. x ∈_ι {r} = (x = r)"
  by(simp)
```

(Θ) (THEOREM) 3.3.10. (Singleton uniqueness)

```
theorem singleton‹uniqueness^N:
  "(∀x. x ∈_ι r ⟷ x = p) ∧ (∀x. x ∈_ι s ⟷ x = p) ⟶ r = s"
  by(metis Ax‹x^N)

theorem
  "∀p.∀r.∀s.(∀x. x ∈_ι r ⟷ x = p) ∧ (∀x. x ∈_ι s ⟷ x = p) ⟶ r = s"
  by(metis Ax‹x^N)
```

(Θ) (THEOREM) 3.3.11. (siS is unique)

```
theorem S_Si‹is‹unique^C:
  "(∀x. x ∈_ι r ⟷ x = s) ⟷ r = {s}"
  by(metis
    S_Si_def
    S_Pa‹is‹unique^C)

theorem
  "∀r.∀s.(∀x. x ∈_ι r ⟷ x = s) ⟷ r = {s}"
  by(metis S_Si‹is‹unique^C)
```

(Θ) (THEOREM) 3.3.12. (siS is a pair)

```
theorem S_Si‹is‹a‹pair^C [simp]:
  "{r,r} = {r}"
   by(metis S_Si_def)

theorem
  "∀r.{r,r} = {r}"
  by(simp)
```

(Θ) (THEOREM) 3.3.13. (emS equals {r} is false)

```
theorem S_Em'‹NEQ‹_S_Em_^C [simp]:
  "(∅ = {r}) = False"
  by(metis
    S_Em‹is‹unique^C
    S_Si‹is‹unique^C)
```

3.3.14 (Unordered Pairs Are Unordered, Element Matching)

$\left(\Theta\right)$ (Theorem) 3.3.15. (paS is unordered)

```
theorem S_Pa'is'unordered^C [simp]:
  "{r,s} = {s,r}"
  by(metis S_Pa'is'unique^C)

theorem
  "∀r.∀s.({r,s} = {s,r})"
  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.

$\left(\kappa\omega\right)$ (Corollary) 3.3.16. (paS {r,s} {p} = paS {p} {r,s})

```
corollary S_Pa'S_Pa'r's'S_Si'p''EQ''S_Pa'S_Si'p'S_Pa'r's^C:
  "{{r,s},{p}} = {{p},{r,s}}"
  by(simp)

corollary
  "∀p.∀r.∀s. {{r,s},{p}} = {{p},{r,s}}"
  by(simp)
```

$\left(\Theta\right)$ (Theorem) 3.3.17. (paS element match)

```
theorem S_Pa'or'element'match^C :
  "{r1,r2} = {s1,s2} ⟷ (r1 = s1 ∧ r2 = s2) ∨ (r1 = s2 ∧ r2 = s1)"
  by(metis Ax'pa^C S_Pa'is'unordered^C)

theorem
  "∀r1.∀r2.∀s1.∀s2.({r1,r2} = {s1,s2} ⟷ (r1 = s1 ∧ r2 = s2) ∨
                                         (r1 = s2 ∧ r2 = s1))"
  by(metis S_Pa'or'element'match^C)
```

3.4 Ordered Pairs

3.4.1 (Ordered Pair Definition, Notation, Existence)

The notation for S_{Op} is \<langle>a,b\<rangle>.

$\left(\nu\nu\right)$ (Notation) 3.4.2. (opS ordered pair)

```
abbreviation (input)
  S_Op :: "σ_ι ⇒ σ_ι ⇒ σ_ι" where
  "S_Op r s == {{r},{r,s}}"
```

```
notation
  S_Op ("opS")

syntax "_S_Op" :: "σ_ι ⇒ σ_ι ⇒ σ_ι" ("(⟨(_,_)⟩)")
  translations
  "⟨r,s⟩" == "CONST S_Op r s"
```

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

(Θ) (Theorem) 3.4.3. (Ordered pairs exist)

```
theorem opairs·exist^C:
  "∃u.(∀x. x ∈_ι u ⟷ x = {r} ∨ x = {r,s})"
  by(metis S_Pa·is·unique^C)

theorem
  "∀r.∀s.∃u.(∀x. x ∈_ι u ⟷ x = {r} ∨ x = {r,s})"
  by(metis opairs·exist^C)
```

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)

```
theorem S_Op·exists^C:
  "∃u.(u = ⟨r,s⟩) ∧ (∀x. x ∈_ι u ⟷ x = {r} ∨ x = {r,s})"
  by(simp)

theorem
  "∀r.∀s.∃u.(u = ⟨r,s⟩) ∧ (∀x. x ∈_ι u ⟷ x = {r} ∨ x = {r,s})"
  by(simp)
```

3.4.5 (Ordered Pair Uniqueness)

(Θ) (Theorem) 3.4.6. (Ordered pair uniqueness)

```
theorem opair·uniqueness^C:
  "(∀x. x ∈_ι r_1 ⟷ (x = {p} ∨ x = {p,q})) ∧
   (∀x. x ∈_ι r_2 ⟷ (x = {p} ∨ x = {p,q})) ⟶ r_1 = r_2"
  by(metis Ax·x^N)

theorem
  "∀p.∀q.∀r_1.∀r_2.(∀x. x ∈_ι r_1 ⟷ (x = {p} ∨ x = {p,q})) ∧
                 (∀x. x ∈_ι r_2 ⟷ (x = {p} ∨ x = {p,q})) ⟶ r_1 = r_2"
  by(metis Ax·x^N)
```

(Θ) (Theorem) 3.4.7. (opS is unique)

```
theorem S_Op·is·unique^C:
  "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q})) ⟷ r = ⟨p,q⟩"
proof assume
  "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q}))"
  thus "r = ⟨p,q⟩"
--"▹By S_Op·EXISTS^C, there exists a set u = ⟨p,q⟩ which has the same
    properties as r. Consequently, because r and u will contain the
    same elements, then by Ax·x^N they are equal.'"
  by(metis
    Ax·x^N S_Op·exists^C)
next assume
  "r = ⟨p,q⟩"
  thus "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q}))"
--"▹Again, by S_Op·EXISTS^C, some u = ⟨p,q⟩ exists, and u has the properties
    of r expressed in the conclusion. By transitivity, r = u, so the
    conclusion holds.'"
  by(metis
    S_Op·exists^C)
qed

theorem
  "∀p.∀q.∀r.(∀x. x ∈_ι r ⟷ (x = {p} ∨ x ={p,q})) ⟷ r = ⟨p,q⟩"
--":"proof fix p show
--":"  "∀q.∀r.(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q})) ⟷ r = ⟨p,q⟩"
--":"proof fix q show
--":"  "∀r.(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q})) ⟷ r = ⟨p,q⟩"
--":"proof fix r show
--":"  "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q})) ⟷ r = ⟨p,q⟩"
proof assume
  "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q}))"
  thus "r = ⟨p,q⟩"
  by(metis
    Ax·x^N S_Op·exists^C)
next assume
  "r = ⟨p,q⟩"
  thus "(∀x. x ∈_ι r ⟷ (x = {p} ∨ x = {p,q}))"
  by(metis
    S_Op·exists^C)
qed qed qed qed
```

3.4.8 (Ordered Pair Element Matching)

(Θ) (Theorem) 3.4.9. (opS element match)

```
theorem S_Op·element·match^C:
  "(⟨r1,r2⟩ = ⟨s1,s2⟩) ⟷ (r1 = s1 ∧ r2 = s2)"
  by(metis
    S_Si_def
    S_Pa·or·element·match^C)

theorem
  "∀r1.∀r2.∀s1.∀s2.(⟨r1,r2⟩ = ⟨s1,s2⟩) ⟷ (r1 = s1 ∧ r2 = s2)"
  by(metis S_Op·element·match^C)
```

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

Implement n-tuples according to [Gol96, 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 $\mathsf{S_{Se}}$ is

- \<lbrace>q\<bar>P\<rbrace> and
- \<lbrace>x\<in>\<^isub>\<iota>\<bar>P\<rbrace>.

$\big(\kappa\tau\big)$ (Constant) 4.1.2. (seS separation: axiomatized by the Axiom of Separation)

```
consts Sₛₑ :: "σι ⇒ (σι ⇒ βι) ⇒ σι"

notation (input)
  Sₛₑ ("seS")

notation
  Sₛₑ ("({(_)|(_)})")
```

$\big(\nu\nu\big)$ (Notation) 4.1.3. (Set builder notation: all x in q such that P)

```

translations "{x ∈ι q | Pₓ}" => "{q|(λx. Pₓ)}"

```

The set q in $\{q \mid P\}$ must be a set which exists, and P should be a function of type $(\sigma_\iota \Rightarrow \beta_\iota)$, as shown by the type of $\mathsf{S_{Se}}$. Additionally, P should be a formula with a free variable. For example, we could have $(P \equiv (\lambda x.\ P_{\mathrm{x}}))$, 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 $x \in_\iota q$ is also true, then by the Axiom of Separation, $x \in_\iota \{q \mid P\}$.

Because P can be any function of type $(\sigma_\iota \Rightarrow \mathtt{bool})$, there is the question of whether a recursive trick can be played to get $\{q \mid P\} \notin_\iota \{q \mid P\}$ 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.

$\big(\alpha\xi\big)$ (Axiom) 4.1.4. (Axiom of Separation: separation sets)

```
axiomatization where
  Ax⋅seᶜ [simp]: "(x ∈ι {q|P}) = (x ∈ι q ∧ P x)"

theorem
  "∀q.∀P.(∀x. x ∈ι {q|P} = (x ∈ι q ∧ P x))"
  by(metis Ax⋅seᶜ)
```

$\big(\alpha\xi\big)$ (Axiom) 4.1.5. (Axiom of Separation: no constant form)

```
theorem  Ax·seᴺ: "∃u.(∀x. x ∈ι u ⟷ (x ∈ι q ∧ P x))"
  by(metis Ax·seᶜ)

theorem
  "∀q.∀P.∃u.(∀x. x ∈ι u ⟷ (x ∈ι q ∧ P x))"
  by(metis Ax·seᶜ)
```

4.1.6 (Separation Set Builder Notation)

As can be seen from Notation 4.1.3, the notation $\{x \in_\iota q \mid P_{\mathrm{x}}\}$ is mapped to $\{q \mid (\lambda x.\ P_{\mathrm{x}})\}$, and so it is equivalent to $\{q \mid P\}$ with $P \equiv (\lambda x.\ P_{\mathrm{x}})$. The formula P_{x} in $\{x \in_\iota q \mid P_{\mathrm{x}}\}$ should be a FOL formula with a free variable x. The practical difference between $\{q \mid P\}$ and $\{x \in_\iota q \mid P_{\mathrm{x}}\}$ is that in $\{x \in_\iota q \mid P_{\mathrm{x}}\}$, 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 $\{x \in_\iota q \mid P_{\mathrm{x}}\}$ is also the desired free variable in P_{x}. Additionally, $\Rightarrow$ cannot be used in place of => in the `translations` command.)

As an example of how to used the notation, and how it relates to the Axiom of Separation, in $\{x \in_\iota q \mid P_{\mathrm{x}}\}$, let $P_{\mathrm{x}} \equiv x \neq x$, then

$$\{x \in_\iota q \mid P_{\mathrm{x}}\} = \{x \in_\iota q \mid x \neq x\} = \{q \mid (\lambda x.\ x \neq x)\}\,. \tag{4.1.1}$$

It can be seen that for the translation

$$\{q \mid (\lambda x.\ x \neq x)\} => \{q \mid P\}\,, \tag{4.1.2}$$

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

$$\forall q.(\forall z.z \in_\iota \{q \mid (\lambda x.\ x \neq x)\} \longleftrightarrow z \in_\iota q \wedge (\lambda x.\ x \neq x)\ z). \tag{4.1.3}$$

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

If there is no free variable x in P_{x}, then unexpected behavior may result, especially if a theorem using $\{x \in_\iota q \mid P_{\mathrm{x}}\}$ 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.

(Θ) (Theorem) 4.1.7. (A lambda calculus equivalency useful for separation sets)

```
theorem
  "P = (λz. P z)"
  by(simp)
```

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

(Θ) (Theorem) 4.1.8. (`seS` equivalent notation)

```
theorem
  "(x ∈ι q ∧ P x ⟶ x ∈ι {q|P}) ∧
   (x ∈ι {q|P} ⟶ x ∈ι {q|(λz. P z)}) ∧
   (x ∈ι {q|(λz. P z)} ⟶ x ∈ι {z ∈ι q | P z}) ∧
```

```
    (x ∈ι {z ∈ι q | P z} ⟶ x ∈ι q ∧ P x)"
  by(metis Ax·se^C)
theorem
  "∀q.∀P.∀x.(x ∈ι q ∧ P x ⟶ x ∈ι {q|P}) ∧
             (x ∈ι {q|P} ⟶ x ∈ι {q | (λz. P z)}) ∧
             (x ∈ι {q|(λz. P z)} ⟶ x ∈ι {z ∈ι q | P z}) ∧
             (x ∈ι {z ∈ι q | P z} ⟶ x ∈ι q ∧ P x)"
  by(metis Ax·se^C)
```

4.1.9 (Separation Set Existence and Uniqueness)

(Θ) (Theorem) 4.1.10. (Separation set uniqueness)

```
theorem sep·set·uniqueness^N:
  "(∀x. x ∈ι r1 ⟷ (x ∈ι q ∧ P x)) ∧
      (∀x. x ∈ι r2 ⟷ (x ∈ι q ∧ P x)) ⟶ r1 = r2"
  by(metis Ax·x^N)

theorem
  "∀q.∀P.∀r1.∀r2.(∀x. x ∈ι r1 ⟷ (x ∈ι q ∧ P x)) ∧
                  (∀x. x ∈ι r2 ⟷ (x ∈ι q ∧ P x)) ⟶ r1 = r2"
  by(metis Ax·x^N)
```

(Θ) (Theorem) 4.1.11. (seS is unique)

```
theorem S_Se·is·unique^C:
  "(∀x. x ∈ι r ⟷ (x ∈ι q ∧ P x)) ⟷ r = {q|P}"
  by(metis Ax·se^C Ax·x^N)

theorem
  "∀q.∀P.∀r.(∀x. x ∈ι r ⟷ (x ∈ι q ∧ P x)) ⟷ r = {q|P}"
  by(metis Ax·x^N Ax·se^C)
```

4.1.12 (Basic Examples Using Set Builder Notation)

(ξπ) (Example) 4.1.13. (emS equals all x not equal to x)

```
theorem
  "∀r. ∅ = {x ∈ι r | x ≠ x}"
  by(metis Ax·em^C S_Se·is·unique^C)
```

(ξπ) (Example) 4.1.14. (The set p is not in the set not containing p)

```
theorem
  "∀r. ∀s. s ∉ι {x ∈ι r | x ≠ s}"
  by(metis (full_types) Ax·se^C)
```

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

```
theorem
  "x ∈ι {r} ⟷ x ∈ι {z ∈ι {r} | z = r}"
  by(metis (full_types)
    Ax›se^C
    S_Si›exists^C)
```

$\left(\xi\pi\right)$ (Example) 4.1.16. (Singletons equal their seS)

```
theorem
  "∀r.({r} = {x ∈ι {r} | x = r})"
  by(metis (full_types)
    S_Si›is›unique^C
    S_Se›is›unique^C)
```

$\left(\xi\pi\right)$ (Example) 4.1.17. (The set containing emS equals its seS)

```
theorem
  "{∅} = {x ∈ι {∅} | x = ∅}"
  by(metis (full_types)
    S_Si›is›unique^C
    S_Se›is›unique^C)
```

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 $\{x \in_\iota q \mid P_{\mathrm{x}}\}$, there is no free variable x. For example, suppose the following equation is used:

$$\{\emptyset\} = \{x \in_\iota \{\emptyset\} \mid y = \emptyset\}\,. \tag{4.1.4}$$

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

$$\forall y.\,\{\emptyset\} = \{x \in_\iota \{\emptyset\} \mid y = \emptyset\}\,. \tag{4.1.5}$$

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:

$$\neg(\{\emptyset\} = \{x \in_\iota \{\emptyset\} \mid y = \emptyset\}) \longleftrightarrow \forall y.\neg(\{\emptyset\} = \{x \in_\iota \{\emptyset\} \mid y = \emptyset\}). \tag{4.1.6}$$

$\left(\xi\pi\right)$ (Example) 4.1.19. (When no free x is in P_{x} for set builder notation)

```
theorem
  "¬(∀r.({∅} = {x ∈ι {∅} | r = ∅}))"
  by(metis (lifting, full_types)
    Ax›em^C
    S_Si›exists^C
    S_Se›is›unique^C)
```

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 $\mathtt{U_{Ge}}$ is `\<^bold>\<Union>`

$\left(\kappa\tau\right)$ (Constant) 4.2.2. (`geU` general union: axiomatized by the Axiom of Unions)

```
consts U_Ge :: "σ_ι ⇒ σ_ι"

notation
  U_Ge ("geU") and
  U_Ge ("⋃")
```

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.

$\left(\alpha\xi\right)$ (Axiom) 4.2.3. (Axiom of Unions)

```
axiomatization where
  Ax·un^C : "x ∈_ι ⋃r = (∃u. x ∈_ι u ∧ u ∈_ι r)"

theorem
  "∀r.∀x.(x ∈_ι ⋃r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι r))"
  by(metis Ax·un^C)
```

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

```
theorem Ax·un^N:
  "∃u.∀x.(x ∈_ι u ⟷ (∃v. x ∈_ι v ∧ v ∈_ι r))"
  by(metis Ax·un^C)

theorem
  "∀r.∃u.∀x.(x ∈_ι u ⟷ (∃v. x ∈_ι v ∧ v ∈_ι r))"
  by(metis Ax·un^C)
```

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:

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

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 "{⋃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 $r = s$.

(Θ) (Theorem) 4.2.8. (Union uniqueness)

```
theorem union·uniquenessᴺ:
  "(∀x. x ∈ι r₁ ⟷ (∃u. x ∈ι u ∧ u ∈ι p)) ∧
    (∀x. x ∈ι r₂ ⟷ (∃u. x ∈ι u ∧ u ∈ι p)) ⟶ r₁ = r₂"
  by(metis Ax·xᴺ)
```

```
theorem
  "∀p.∀r_1.∀r_2.(∀x. x ∈_ι r_1 ⟷ (∃u. x ∈_ι u ∧ u ∈_ι p)) ∧
                (∀x. x ∈_ι r_2 ⟷ (∃u. x ∈_ι u ∧ u ∈_ι p)) ⟶ r_1 = r_2"
  by(metis Ax·x^N)
```

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

(Θ) (Theorem) 4.2.9. (geU is unique)

```
theorem U_Ge·is·unique^C:
  "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s)) ⟷ r = ⋃s"
proof assume
  "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s))"
  thus "r = ⋃s"
--"▹By Ax·un^C, ⋃s has the properties of r in the hypothesis, therefore by
    Ax·x^N, r = ⋃s.'"
  by(metis
    Ax·x^N
    Ax·un^C)
next assume
  "r = ⋃s"
  thus "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s))"
--"▹Let P be the formula stated in the conclusion, then by axiom Hol.subst,
    (λr.P)r = (λr.P)⋃p. Because (λr.P)⋃p is true by Ax·un^C, then the
    conclusion follows.'"
  by(metis
    Ax·un^C)
qed

theorem
  "∀r.∀s.(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s)) ⟷ r = ⋃s"
--":"proof fix r show
--":"  "∀s.(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s)) ⟷ r = ⋃s"
--":"proof fix s show
--":"  "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s)) ⟷ r = ⋃s"
proof assume
  "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s))"
  thus "r = ⋃s"
  by(metis
    Ax·un^C Ax·x^N)
next assume
  "r = ⋃s"
  thus "(∀x. x ∈_ι r ⟷ (∃u. x ∈_ι u ∧ u ∈_ι s))"
  by(metis
    Ax·un^C)
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 $\mathtt{S}_{\mathtt{Pa}}$ and $\mathtt{U}_{\mathtt{Ge}}$ are the building blocks for $\mathtt{U}_{\mathtt{Bi}}$, $\mathtt{U}_{\mathtt{Fi}}$, and $\mathtt{S}_{\mathtt{Fi}}$, which are binary union, finite union, and finite set respectively, then theorems need to be proved about $\mathtt{U}_{\mathtt{Ge}}$ unions using $\mathtt{S}_{\mathtt{Pa}}$.

(Θ) (Theorem) 4.2.11. (Unions of paS exist)

```
theorem unions'of'S_pa'exist^C:
  "∃u.∀x.(x ∈_ι u ⟷ (∃v. x ∈_ι v ∧ v ∈_ι {r,s}))"
  by(metis Ax'un^C)

theorem
  "∀r.∀s.∃u.∀x.(x ∈_ι u ⟷ (∃v. x ∈_ι v ∧ v ∈_ι {r,s}))"
  by(metis unions'of'S_pa'exist^C)
```

(Θ) (Theorem) 4.2.12. (geU paS exists)

```
theorem U_Ge'S_Pa'exists^C:
  "x ∈_ι ⋃{r,s} ⟷ (∃u. x ∈_ι u ∧ u ∈_ι {r,s})"
  by(metis
    U_Ge'is'unique^C)

theorem
  "∀r.∀s.∀x. x ∈_ι ⋃{r,s} ⟷ (∃u. x ∈_ι u ∧ u ∈_ι {r,s})"
  by(metis U_Ge'S_Pa'exists^C)
```

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.

(Θ) (Theorem) 4.2.13. (geU siS equals inP)

```
theorem U_Ge'S_Si''EQ''P_In^C [simp]:
  "x ∈_ι ⋃{r} = x ∈_ι r"
  by(metis
    Ax'un^C
    S_Si'is'unique^C)

theorem
  "∀r.∀x. x ∈_ι ⋃{r} = x ∈_ι r"
  by(metis U_Ge'S_Si''EQ''P_In^C)
```

(Θ) (Theorem) 4.2.14. (geU paS equals inP or)

```
theorem U_Ge'S_Pa''EQ''P_In'or^C [simp]:
  "x ∈_ι ⋃{r,s} = (x ∈_ι r ∨ x ∈_ι s)"
  by(metis
    Ax'pa^C
    U_Ge'S_Pa'exists^C)

theorem
  "∀r.∀s.∀x. x ∈_ι ⋃{r,s} = (x ∈_ι r ∨ x ∈_ι s)"
  by(metis U_Ge'S_Pa''EQ''P_In'or^C)
```

(Θ) (Theorem) 4.2.15. (geU{r} = r)

```
theorem U_Ge_r_'EQ''r^C [simp]:
  "⋃{r} = r"
proof- have
  "⋃{r} ⊆_ϵ r"
```

```
  by(simp)
  thus
  "⋃{r} = r"
  by(metis 0_Ss'eq^N)
qed

theorem "∀r.⋃{r} = r"
  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

$$\bigcup\left\{\bigcup\{p,r\},\bigcup\{p,s\}\right\}, \tag{4.2.2}$$

after 4.2.18 is introduced, will return the value

$$\bigcup\left\{p,\bigcup\{r,\bigcup\{p,s\}\}\right\}. \tag{4.2.3}$$

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)

```
theorem U_Ge'p'into'U_Ge'S_Pa^C [simp]:
  "⋃{⋃{p,r},⋃{p,s}} = ⋃{p,⋃{r,s}}"
proof- have
  "⋃{⋃{p,r},⋃{p,s}} ⊆_ϵ ⋃{p,⋃{r,s}}"
  by(simp)
  thus
  "⋃{⋃{p,r},⋃{p,s}} = ⋃{p,⋃{r,s}}"
  by(metis
    0_Ss'eq^N)
qed

theorem
  "∀p.∀r.∀s.⋃{⋃{p,r},⋃{p,s}} = ⋃{p,⋃{r,s}}"
  by(metis U_Ge'p'into'U_Ge'S_Pa^C)
```

(Θ) (Theorem) 4.2.18. (geU is associative)

```
theorem U_Ge'is'associative^C [simp]:
  "⋃{⋃{p,r},s} = ⋃{p,⋃{r,s}}"
proof- have
  "⋃{⋃{p,r},s} ⊆_ϵ ⋃{p,⋃{r,s}}"
  by(simp)
  thus
  "⋃{⋃{p,r},s} = ⋃{p,⋃{r,s}}"
  by(metis
```

```
    O_Ss·eq^N)
qed

theorem
  "∀p.∀r.∀s.⋃{⋃{p,r},s} = ⋃{p,⋃{r,s}}"
  by(metis U_Ge·is·associative^C)
```

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)

```
theorem U_Ge·left·commute^C [simp]:
  "⋃{p,⋃{r,s}} = ⋃{r,⋃{p,s}}"
   by(metis
     S_Pa·is·unordered^C
     U_Ge·is·associative^C)

theorem
  "∀p.∀r.∀s.⋃{p,⋃{r,s}} = ⋃{r,⋃{p,s}}"
  by(simp)
```

4.2.20 (Pseudo Associate and Commute)

(Θ) (Theorem) 4.2.21. (`geU{siS,paS}` left commute)

```
theorem U_Ge_S_Si-S_Pa_left·commute^C [simp]:
  "⋃{{p},{r,s}} = ⋃{{r},{p,s}}"
proof-
  have
  "⋃{{p},{r,s}} ⊆_ϵ ⋃{{r},{p,s}}"
  by(simp)
  thus
  "⋃{{p},{r,s}} = ⋃{{r},{p,s}}"
  by(metis
    O_Ss·eq^N)
qed

theorem
  "⋃{{p},{r,s}} = ⋃{{r},{p,s}}"
  by(simp)
```

(Θ) (Theorem) 4.2.22. (`geU{paS,paS}` inside commute)

```
theorem U_Ge_S_Pa-S_Pa_inside·commute^C [simp]:
  "⋃{{p,q},{r,s}} = ⋃{{p,r},{q,s}}"
proof-
  have
  "⋃{{p,q},{r,s}} ⊆_ϵ ⋃{{p,r},{q,s}}"
  by(simp)
  thus
  "⋃{{p,q},{r,s}} = ⋃{{p,r},{q,s}}"
  by(metis
    O_Ss·eq^N)
```

```
qed

theorem
  "∀p.∀q.∀r.∀s. ⋃{{p,q},{r,s}} = ⋃{{p,r},{q,s}}"
  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

$$\bigcup\left\{\left\{p,\bigcup q\right\},\{\{k,\{l\}\}\}\right\} \equiv \bigcup\left\{\{p\},\left\{\bigcup q,\{k,\{l\}\}\right\}\right\}. \tag{4.2.4}$$

(Θ) (Theorem) 4.2.23. (`geU{paS,siS}` associates)

```
theorem Uge_SPa-SSi_associatesC [simp]:
  "⋃{{p,r},{s}} = ⋃{{p},{r,s}}"
  by(simp)

theorem
  "∀p.∀r.∀s.⋃{{p,r},{s}} = ⋃{{p},{r,s}}"
  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.

(Θ) (Theorem) 4.2.25. (`geU{paS,paS}` twin isolate)

```
theorem UGe_SPa-SPa_twin·isolateC [simp]:
  "⋃{{p,r},{r,s}} = ⋃{{r},{p,s}}"
  by(metis
    SSi·is·a·pairC
    SPa·is·unorderedC
    UGe_SPa-SPa_inside·commuteC)

theorem
  "∀p.∀r.∀s.⋃{{p,r},{r,s}} = ⋃{{r},{p,s}}"
  by(simp)
```

(Θ) (Theorem) 4.2.26. (`geU emS = emS`)

```
theorem UGe·SEm··EQ··SEmC [simp]:
  "⋃∅ = ∅"
  by(metis
    Ax·emC
    UGe·is·uniqueC)
```

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

```
theorem UGe_SEm-r_›EQ››rC [simp]:
  "⋃{∅,r} = r"
proof- have
  "⋃{∅,r} ⊆ϵ r"
  by(simp)
  thus
  "⋃{∅,r} = r"
  by(metis
    0Ss›eqN)
qed

theorem
  "∀r.⋃{∅,r} = r"
  by(metis UGe_SEm-r_›EQ››rC)
```

4.2.28 (Eliminating All Unions)

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

```
theorem UGe_SSi-SSi_›EQ››SPaC [simp]:
  "⋃{{r},{s}} = {r,s}"
  by(metis
    UGe_r_›EQ››rC
    SSi›is›a›pairC
    UGe_SPa-SPa_inside›commuteC)

theorem
  "∀r.∀s.⋃{{r},{s}} = ⋃{{r,s}}"
  by(simp)
```

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

```
theorem UGe__r_-_r-s__EQ›_r-s_C [simp]:
  "⋃{{r},{r,s}} = {r,s}"
   by(metis
     SSi›is›a›pairC
     SPa›is›unorderedC
     UGe_SSi-SSi_›EQ››SPaC
     UGe_SSi-SPa_left›commuteC)

theorem
  "∀r.∀s.⋃{{r},{r,s}} = {r,s}"
  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 σ_ι.

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

```
theorem U_Ge'U_Ge_r-_s__EQ''U_Ge_s-U_Ge'r_^C [simp]:
  "⋃(⋃{r,{s}}) = ⋃{s,⋃r}"
proof- have
  "∀x. x ∈ι ⋃(⋃{r,{s}}) ⟷ (∃u. x ∈ι u ∧ u ∈ι ⋃{r,{s}})"
  by(smt
    Ax'un^C)
  hence
  "⋃(⋃{r,{s}}) ⊆ϵ ⋃{s,⋃r}"
  apply(simp)
  by(metis
    Ax'un^C)
  thus
  "⋃(⋃{r,{s}}) = ⋃{s,⋃r}"
  by(metis
    O_Ss'eq^N)
qed

theorem
  "∀r.∀s.⋃(⋃{s,{r}}) = ⋃{r,⋃s}"
  by(simp)
```

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

$\left(\kappa\omega\right)$ (Corollary) 4.2.33. (geU geU{{r},s} = geU{r,geU s})

```
corollary U_Ge'U_Ge__r_-s_EQ''U_Ge_r-U_Ge's_^C [simp]:
  "⋃(⋃{{r},s}) = ⋃{r,⋃s}"
  by(simp)

corollary
  "∀r.∀s.⋃(⋃{{r},s}) = ⋃{r,⋃s}"
  by(simp)
```

A pair is expanded into singletons, since `fiS` is a union of singletons.

$\left(\Theta\right)$ (Theorem) 4.2.34. (geU{{p,r},s} = geU{{p},geU{{r},s}})

```
theorem U_Ge__p-r_-s_'EQ''U_Ge__p_-U_Ge__r_-s__^C [simp]:
  "⋃{{p,r},s} = ⋃{{p},⋃{{r},s}}"
  by(simp)

theorem
  "∀p.∀r.∀s.⋃{{p,r},s} = ⋃{{p},⋃{{r},s}}"
  by(simp)
```

4.2.35 (Eliminating Some Unions)

$\left(\Theta\right)$ (Theorem) 4.2.36. (geU{r,geU{r,s}} = geU{r,s})

```
theorem U_Ge_r-U_Ge_r-s__EQ''U_Ge_r-s_^C [simp]:
  "⋃{r,⋃{r,s}} = ⋃{r,s}"
  by(metis
    Ax'x^N
```

```
    U_Ge'S_Pa''EQ''P_In'or^C)

theorem
  "⋃{r,⋃{r,s}} = ⋃{r,s}"
  by(simp)
```

(Θ) (Theorem) 4.2.37. (geU{r,geU{s}} = geU{r,s})

```
theorem U_Ge_r-U_Ge_s__EQ''U_Ge_r-s_^C [simp]:
  "⋃{r,⋃{s}} = ⋃{r,s}"
  by(metis
    U_Ge_S_Si-S_Si_'EQ''S_Pa^C
    U_Ge'U_Ge__r_-s_EQ''U_Ge_r-U_Ge's_^C)

theorem
  "∀r.∀s. ⋃{r,⋃{s}} = ⋃{r,s}"
  by(simp)
```

(κω) (Corollary) 4.2.38. (geU{geU{r},geU{s}} = geU{r,s})

```
corollary U_Ge_U_Ge_r-U_Ge_s__EQ''U_Ge_r-s_^C [simp]:
  "⋃{⋃{r},⋃{s}} = ⋃{r,s}"
  by(simp)

corollary
  "∀r.∀s.⋃{⋃{r},⋃{s}} = ⋃{r,s}"
  by(simp)
```

(Θ) (Theorem) 4.2.39. (geU{geU{r,s}} = geu{r,s})

```
theorem U_Ge_U_Ge_r-s__EQ''U_Ge_r-s_^C [simp]:
  "⋃{⋃{r,s}} = ⋃{r,s}"
  by(metis
    S_Si'is'a'pair^C
    S_Pa'is'unordered^C
    U_Ge_r-U_Ge_r-s__EQ''U_Ge_r-s_^C)

theorem
  "∀r.∀s.⋃{⋃{r,s}} = ⋃{r,s}"
  by(simp)
```

4.2.40 (Miscellaneous Results)

(Θ) (Theorem) 4.2.41. (p in r implies p a subset of geU r)

```
theorem p'in'r''IMP''p'a'subset'of'U_Ge'r^C:
  "p ∈_ι r ⟶ p ⊂_ι ⋃r"
  by(metis Ax'un^C O_SS_def)

theorem
  "∀p.∀r. p ∈_ι r ⟶ p ⊂_ι ⋃r"
  by(metis Ax'un^C O_SS_def)
```

(Θ) (Theorem) 4.2.42. (geU{x,{x}} exists)

```
theorem U_Ge_x-_x__exists^C:
  "∃u. u = ⋃{x,{x}}"
  by(metis Ax·un^C)

theorem
  "∀x.∃u. u = ⋃{x,{x}}"
  by(metis Ax·un^C)
```

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 $\cup_\iota$ 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 \<^bold>\<union>

(νν) (Notation) 4.2.44. (biU Binary Union)

```
abbreviation (input)
  U_Bi :: "σ_ι ⇒ σ_ι ⇒ σ_ι" where "U_Bi r s == ⋃{r,s}"

notation
  U_Bi  ("bi'_U") and
  U_Bi  (infixl "biU" 65) and
  U_Bi  (infixl "∪_ι" 65)

--"Undefined notation."
--"⋃{}"
```

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

(κω) (Corollary) 4.2.45. (biU commutes)

```
corollary
  "r ∪_ι s = s ∪_ι r"
  by(simp)

corollary
  "∀r.∀s. r ∪_ι s = s ∪_ι r"
  by(simp)
```

(κω) (Corollary) 4.2.46. (biU left commutes)

```
corollary
  "p ∪_ι (r ∪_ι s) = r ∪_ι (p ∪_ι s)"
  by(simp)

corollary
  "∀p.∀r.∀s. p ∪_ι (r ∪_ι s) = r ∪_ι (p ∪_ι s)"
  by(simp)
```

$\left(\kappa\omega\right)$ (Corollary) 4.2.47. (biU is associative)

```
corollary
  "(p ∪ι r) ∪ι s = p ∪ι (r ∪ι s)"
  by(simp)

corollary
  "∀p.∀r.∀s.(p ∪ι r) ∪ι s = p ∪ι (r ∪ι s)"
  by(simp)
```

$\left(\kappa\omega\right)$ (Corollary) 4.2.48. (biU distribute into union)

```
corollary
  "p ∪ι (r ∪ι s) = (p ∪ι r) ∪ι (p ∪ι s)"
  by(simp)

corollary
  "∀p.∀r.∀s. p ∪ι (r ∪ι s) = (p ∪ι r) ∪ι (p ∪ι s)"
  by(simp)
```

4.3 Finite Sets

4.3.1 (Finite set fiS, Equality Examples)

The notation for S_{Fi} is `\<lbrace>`...`\<rbrace>`.

$\left(\rho\delta\right)$ (Rdefinition) 4.3.2. (fiS finite set)

```
fun SFi :: "σ∧ ⇒ σι" where
  "SFi []      = ∅"
| "SFi (r#rs) = (⋃{{r},SFi rs})"

notation (input)
  SFi  ("fiS")

syntax "_SFi" :: "σι ⇒ σι⇒ args ⇒ σι" ("({(_,_,_)})")

translations
  "{r,s,ss}" == "CONST SFi (r#s#[ss])"
```

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

```
theorem "⋃{{a,b},{c}} = {a,b,c} ∧
             (x ∈ι {a,b,c} ⟷ x = a ∨ x = b ∨ x = c)"
  by(simp)

value "⋃{{a,b},{c}}"
      --"⋃{{a,b},{c,c}}"
value "{a,b,c}"
      --"⋃{{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)

```
theorem "{a,z,f,g,m,f,t} = {z,g,m,z,t,a,f}"
  by(simp)

theorem "{z,a,{k,{l}}} = {{k,{l}},z,{k,{l}},a}"
  by(simp)

theorem "{k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{s,t,s,t,t}}}}}}}}}}
         ∈ᵢ
         {{a,b,{m}},{a,b,{m}},d,z,b,a,
            {k,k,{l,{m,m,{n,{a,{a,z,{x,{p,{q,{r,{s,s,s,t,s,t}}}}}}}}}}}}"
  by(simp)

theorem "{{a,b,{m}},{a,b,{m}},d,z,b,a,
            {k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{t,s,t}}}}}}}}}}}}
         =
         {{k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{s,t}}}}}}}}}}},a,z,b,
            {k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{t,s}}}}}}}}}}},{a,{m},b},d}"
  by(simp)

theorem "{{a,b,{m}},{a,b,{m}},d,z,b,a,
            {k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{t,s,t}}}}}}}}}}}}
         =
         {{k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{s,t}}}}}}}}}}},a,z,b,
            {k,{l,{m,{n,{a,{a,z,{x,{p,{q,{r,{t,s}}}}}}}}}}},{a,{m},b},d}"
  apply(simp del:
    S_Si'is'a'pair^C
    U_Ge_S_Em-r_'EQ''r^C
    S_Pa'is'unordered^C
    U_Ge'left'commute^C
    U_Ge_S_Si-S_Si_'EQ''S_Pa^C
    U_Ge__r_-_r-s__EQ'_r-s_^C
    U_ge_S_Pa-S_Si_associates^C
    U_Ge_S_Si-S_Pa_left'commute^C)
  by(simp)
```

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

```
theorem
  "⋃{p,⋃{r,s}} = ⋃{p,r,s}"
  by(simp)

theorem
  "⋃{p,⋃{q,⋃{r,s}}} = ⋃{p,q,r,s}"
  by(simp)

theorem
  "{a,b,a,d} ∪ι {a,c,m,c} = {m,d,c,b,a}"
  by(simp)

theorem
  "⋃{{a,b,c,d,c},{a,m,{n},p}} = {a,b,c,d,m,{n},p}"
  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.

(Θ) (Theorem) 4.3.7. (fiS([r] @ rs) = fiS(r # rs))

```
theorem SFi‚’r’‚rs‚‚EQ‚‚SFi‚r‚rsᶜ:
  "SFi([r] @ rs) = SFi(r # rs)"
  by(simp)

theorem
  "∀r.∀rs. SFi([r] @ rs) = SFi(r # rs)"
  by(simp)
```

Likewise for rs@[r], where the right-hand side of 4.3.8 simplifies to $\bigcup\{S_{\mathrm{Fi}}\ rs, \{r\}\}$, 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.

(Θ) (Theorem) 4.3.8. (fiS(rs @ [r]) = fiS(r # rs))

```
theorem SFi‚rs‚r‚‚EQ‚‚SFi‚r‚rsᶜ:
  "SFi(rs @ [r]) = SFi(r # rs)"
  apply(induction rs arbitrary: r)
  by(auto)

theorem
  "∀r.∀rs. SFi(rs @ [r]) = SFi(r # rs)"
  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.

(Θ) (Theorem) 4.3.9. (geU(fiS(r # rs)) = geU{r,geU(fiS rs))

```
theorem UGe‚SFi‚r‚rs‚‚EQ‚‚UGe_r-UGe‚SFi‚rs_ᶜ:
  "⋃(SFi(r # rs)) = ⋃{r,⋃(SFi rs)}"
  by(simp)

theorem
  "∀r.∀rs.⋃(SFi(r # rs)) = ⋃{r,⋃(SFi rs)}"
  by(simp)
```

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

(Θ) (Theorem) 4.3.10. (geU(fiS([r] @ rs)) = geU{r,geU(fiS rs))

```
theorem UGe·SFi·'r'·rs··EQ··UGe_r-UGe·SFi·rs_C:
  "⋃(SFi([r] @ rs)) = ⋃{r,⋃(SFi rs)}"
  by(simp)

theorem
  "∀r.∀rs.⋃(SFi([r] @ rs)) = ⋃{r,⋃(SFi rs)}"
  by(simp)
```

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

(Θ) (Theorem) 4.3.12. (geU{fiS} = fiS)

```
theorem UGe_SFi_·EQ··SFiC [simp]:
  "⋃{SFi rs} = SFi rs"
  apply(induction rs)
  by(auto)

theorem
  "∀rs.⋃{SFi rs} = SFi rs"
  by(simp)
```

(Θ) (Theorem) 4.3.13. (fiS append to pair)

```
theorem SFi·append·to·pairC [simp]:
  "SFi(rs @ ss) = ⋃{SFi rs,SFi ss}"
  apply(induction rs)
  by(auto)

theorem
  "∀rs.∀ss. SFi(rs @ ss) = ⋃{SFi rs,SFi ss}"
  by(simp)
```

(Θ) (Theorem) 4.3.14. (geU{geU fiS} = geU fiS)

```
theorem UGe_UGe·SFi_·EQ··UGe·SFiC [simp]:
  "⋃{⋃(SFi rs)} = ⋃(SFi rs)"
  apply(induction rs)
  by(auto)

theorem
  "∀rs.⋃{⋃(SFi rs)} = ⋃(SFi rs)"
  by(simp)
```

(Θ) (Theorem) 4.3.15. (geU{geU fiS,geU fiS} = geU(geU{fiS,fiS}))

```
theorem UGe_UGe'SFi-UGe'SFi_'EQ''UGe'UGe_SFi-SFi_C [simp]:
  "⋃{⋃(SFi rs),⋃(SFi ss)} = ⋃(⋃{SFi rs,SFi ss})"
  apply(induction rs arbitrary: ss)
  apply(auto)
  by (metis
    UGe'is'associativeC
    UGe'UGe_r-_s__EQ''UGe_s-UGe'r_C)

theorem
  "∀rs.∀ss.⋃{⋃(SFi rs),⋃(SFi ss)} = ⋃(⋃{SFi rs,SFi ss})"
  by(simp)
```

$\left(\xi\pi\right)$ (Example) 4.3.16. (`fiS` equations solved using `simp` only)

```
theorem --"Associativity."
  "SFi((ps @ rs) @ ss) = SFi(ps @ (rs @ ss))"
  by(simp)

theorem --"Left commute."
  "SFi(ps @ (rs @ ss)) = SFi(rs @ (ps @ ss))"
  by(simp)

theorem --"Inside commute."
  "SFi((ps @ qs) @ (rs @ ss)) = SFi((ps @ rs) @ (qs @ ss))"
  by(simp)

theorem --"Order does not matter, and duplicate Cons and appends do not matter."
  "SFi((ps @ x # qs @ [f,g,c]) @ (rs @ q # ss)@ (rs @ [d,c,d] @ xs)) =
     SFi((((xs @ [q]) @ ps) @ q # qs) @ ([q] @ (x # rs @ [c,d,f,g,c]) @ ss))"
  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.

$\left(\Delta\right)$ (Definition) 4.4.2. (`geI` general intersection)

```
definition IGe :: "σι ⇒ σι" where
  "IGe r = {x ∈ι ⋃r | ∀p. p ∈ι r ⟶ x ∈ι p}"

notation
  IGe ("geI") and
  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.

$\left(\nu\nu\right)$ (Notation) 4.4.3. (`biI` binary intersection)

```
abbreviation (input)
  I_Bi :: "σ_ι ⇒ σ_ι ⇒ σ_ι" where "I_Bi r s == ⋂{r,s}"

notation
  I_Bi  ("bi'_I") and
  I_Bi  (infixl "biI" 70) and
  I_Bi  (infixl "∩_ι" 70)
```

(Θ) (Theorem) 4.4.4. (geI siS equals inP)

```
theorem I_Ge'S_Si''EQ''P_In^C [simp]:
  "x ∈_ι ⋂{r} = (x ∈_ι r)"
  apply(unfold I_Ge_def)
  by(auto)

theorem
  "∀r.∀x. x ∈_ι ⋂{r} = (x ∈_ι r)"
  apply(unfold I_Ge_def)
  by(auto)
```

(Θ) (Theorem) 4.4.5. (geI paS equals inP and)

```
theorem I_Ge'S_Pa''EQ''P_In'and^C [simp]:
  "x ∈_ι ⋂{r,s} = (x ∈_ι r ∧ x ∈_ι s)"
  apply(unfold I_Ge_def)
  by(auto)

theorem
  "∀r.∀s.∀x. x ∈_ι ⋂{r,s} = (x ∈_ι r ∧ x ∈_ι s)"
  apply(unfold I_Ge_def)
  by(auto)
```

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

(Θ) (Theorem) 4.4.7. (NEW: geI is associative)

```
theorem inter'associate [simp]:
  "⋂{⋂{p,r},s} = ⋂{p,⋂{r,s}}"
proof- have
  "⋂{⋂{p,r},s} ⊆_ϵ ⋂{p,⋂{r,s}}"
  by(simp)
  thus
  "⋂{⋂{p,r},s} = ⋂{p,⋂{r,s}}"
  by(metis 0_SS'eq^N)
qed
```

(Θ) (Theorem) 4.4.8. (NEW: geI left commute)

```
theorem inter⋅left⋅commute [simp]:
  "⋂{p,⋂{r,s}} = ⋂{r,⋂{p,s}}"
proof- have
  "⋂{p,⋂{r,s}} ⊆ϵ ⋂{r,⋂{p,s}}"
  by(simp)
  thus
  "⋂{p,⋂{r,s}} = ⋂{r,⋂{p,s}}"
  by(metis 0_Ss⋅eq^N)
qed
```

(Θ) (Theorem) 4.4.9. (geI{r} = r)

```
theorem I_Ge_r_⋅EQ⋅⋅r^C [simp]:
  "⋂{r} = r"
proof- have
  "⋂{r} ⊆ϵ r"
  by(simp)
  thus
  "⋂{r} = r"
  by(metis 0_Ss⋅eq^N)
qed
```

(Θ) (Theorem) 4.4.10. (NEW: geI of two singletons)

```
theorem NEWmbyh40a:
  "⋂{{r},{r}} = {r}"
  by(metis
    I_Ge_r_⋅EQ⋅⋅r^C
    S_Si⋅is⋅a⋅pair^C)
```

4.4.11 (NEW: Distribute)

(Θ) (Theorem) 4.4.12. (NEW: Left distribute)

```
theorem NEWmbyh57b [simp]:
  "⋂{p,⋃{r,s}} = ⋃{⋂{p,r},⋂{p,s}}"
proof- have
  "⋂{p,⋃{r,s}} ⊆ϵ ⋃{⋂{p,r},⋂{p,s}}"
  by(simp)
  thus
  "⋂{p,⋃{r,s}} = ⋃{⋂{p,r},⋂{p,s}}"
  by(metis 0_Ss⋅eq^N)
qed
```

(Θ) (Theorem) 4.4.13. (NEW: Right distribute)

```
theorem NEWmbya34a [simp]:
  "⋂{⋃{p,r},s} = ⋃{⋂{p,s},⋂{r,s}}"
proof- have
  "⋂{⋃{p,r},s} ⊆ϵ ⋃{⋂{p,s},⋂{r,s}}"
```

```
  by(simp)
  thus
  "⋂{⋃{p,r},s} = ⋃{⋂{p,s},⋂{r,s}}"
  by(metis O_Ss,eq^N)
qed
```

(Θ) (Theorem) 4.4.14. (NEW: Left distribute into pair)

```
theorem NEWmbyh59a [simp]:
  "⋂{p,{r,s}} = ⋃{⋂{p,{r}},⋂{p,{s}}}"
proof- have
  "⋂{p,{r,s}} ⊆_ϵ ⋃{⋂{p,{r}},⋂{p,{s}}}"
  by(auto)
  thus
  "⋂{p,{r,s}} = ⋃{⋂{p,{r}},⋂{p,{s}}}"
  by(metis O_Ss,eq^N)
qed
```

(Θ) (Theorem) 4.4.15. (NEW: Right distribute into pair)

```
theorem NEWmbyh60a  [simp]:
  "⋂{{r,s},p} = ⋃{⋂{p,{r}},⋂{p,{s}}}"
  by(simp)
```

4.4.16 (NEW: Example)

(ξπ) (Example) 4.4.17. (NEW: example)

```

theorem stuffer2 [simp]:
  "(r = ⋃{{r},s}) = False"
  sorry
  thm "stuffer2"

theorem stuffer3 [simp]:
  "(r = {s,{r}}) = False"
  sorry
  thm "stuffer3"

--"
a = ∅
b = {∅}
c = {{∅}} or {∅,{∅}}
d = {{{∅}}} or {∅,{∅},{∅,{∅}}}
"

theorem --"equal subsets operator"
  "⋂{{  ∅, {∅}, {∅,{∅}}, {∅,{∅},{∅,{∅}}}  },{  {∅}, {∅,{∅}}, {∅,{∅},{∅,{∅}}}  }}
     ⊆_ϵ { {∅}, {∅,{∅}}, {∅,{∅},{∅,{∅}}} }"
  by(simp)

theorem --"equal subsets operator"
```

```
  "⋂{{  ∅,{∅},{{{∅}}}  },{  {∅},{{∅}},{{{∅}}}  }} ⊆ϵ {{∅},{{{∅}}}}"
  by(simp)

theorem --"equal"
  "⋂{{  ∅,{∅},{{{∅}}}  },{  {∅},{{∅}},{{{∅}}}  }} = {{∅},{{{∅}}}}"
  apply(simp)
  oops

theorem "⋂{{a,b,y,z},{m,n,y,z}} = {z,y}"
  apply simp
  oops

theorem "y ∈ι ⋂{{a,b,y,z},{m,n,y,z}}"
  by simp

theorem "z ∈ι ⋂{{a,b,y,z},{m,n,y,z}}"
  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 \<P>\<^isub>\<S>.

$(\kappa\tau)$ (Constant) 4.5.2. (pwS power set: axiomatized by the Axiom of Power Sets)

```
consts 𝒫S :: "sT ⇒ sT"

notation (input)
  𝒫S ("pwS")
```

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

```
axiomatization where
  Ax·pw^C: "(x ∈ι 𝒫S(r) ⟷ x ⊂ι r)"

theorem
  "∀r.∀x.(x ∈ι 𝒫S(r) ⟷ x ⊂ι r)"
  by(metis Ax·pw^C)
```

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

```
theorem Ax·pw^N:
  "∃u.∀x.(x ∈ι u ⟷ x ⊂ι r)"
  by(metis Ax·pw^C)

theorem
  "∀r.∃u.∀x.(x ∈ι u ⟷ x ⊂ι r)"
  by(metis Ax·pw^C)
```

4.5.5 (Power Set Uniqueness)

(Θ) (Theorem) 4.5.6. (Power set uniqueness)

```
theorem power·set·uniqueness^N:
  "(∀x. x ∈_ι r_1 ⟷ x ⊂_ι p) ∧
     (∀x. x ∈_ι r_2 ⟷ x ⊂_ι p) ⟶ r_1 = r_2"
  by(metis Ax·x^N)

theorem
  "∀p.∀r_1.∀r_2.(∀x. x ∈_ι r_1 ⟷ x ⊂_ι p) ∧
                    (∀x. x ∈_ι r_2 ⟷ x ⊂_ι p) ⟶ r_1 = r_2"
  by(metis Ax·x^N)
```

(Θ) (Theorem) 4.5.7. (pwS is unique)

```
theorem 𝒫_S·is·unique^C:
  "(∀x. x ∈_ι r ⟷ x ⊂_ι s) ⟷ r = 𝒫_S(s)"
proof assume
  "(∀x. x ∈_ι r ⟷ x ⊂_ι s)"
  thus "r = 𝒫_S(s)"
  by(metis
    Ax·pw^C Ax·x^N)
next assume
  "r = 𝒫_S(s)"
  thus "(∀x. x ∈_ι r ⟷ x ⊂_ι s)"
  by(metis
    Ax·pw^C)
qed

theorem
  "∀s.∀r.(∀x. x ∈_ι r ⟷ x ⊂_ι s) ⟷ r = 𝒫_S(s)"
proof fix s show
  "∀r.(∀x. x ∈_ι r ⟷ x ⊂_ι s) ⟷ r = 𝒫_S(s)"
proof fix r show
  "(∀x. x ∈_ι r ⟷ x ⊂_ι s) ⟷ r = 𝒫_S(s)"
proof assume
  "(∀x. x ∈_ι r ⟷ x ⊂_ι s)"
  thus "r = 𝒫_S(s)"
  by(metis
    Ax·pw^C Ax·x^N)
next assume
  "r = 𝒫_S(s)"
  thus "(∀x. x ∈_ι r ⟷ x ⊂_ι s)"
  by(metis
    Ax·pw^C)
qed qed qed
```

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

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

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

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

$\left(\iota\sigma\right)$ (Isar) 6.0.1. (Theory end)

```
end
```

Bibliography

[Aga07] Neithan Agarwaen. Extensions by definitions. Wikipedia, 2007. 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