

## Chapter 4

# ITT Implementation of CZF Theory

The *CZF* theory formalizes Aczel’s constructive set theory [Acz86]. In *Notes on Constructive Set Theory* [Acz97], Aczel provides the central motivation for the set theory.

“The general topic of Constructive Set Theory originated in the seminal 1975 paper of John Myhill, where a specific axiom system CST was introduced. Constructive Set Theory provides a standard set theoretical framework for the development of constructive mathematics in the style of Errett Bishop [Bis67] and is one of several such frameworks for constructive mathematics that have been considered. It is distinctive in that it uses the standard first order language of classical axiomatic set theory and makes no explicit use of specifically constructive ideas. Of course its logic is intuitionistic, but there is no special notion of construction or constructive object. There are just the sets, as in classical set theory. This means that mathematics in constructive set theory can look very much like ordinary classical mathematics. The advantage of this is that the ideas, conventions and practise of the set theoretical presentation of ordinary mathematics can be used also in the set theoretical development of constructive mathematics, provided that a suitable discipline is adhered to. In the first place only the methods of logical reasoning available in intuitionistic logic should be used. In addition only the set theoretical axioms allowed in constructive set theory can be used. With some practise it is not difficult for the constructive mathematician to adhere to this discipline.”

Aczel assures the constructivity of the set theory by using Martin–Löf’s Constructive Set Theory as its foundation. We follow Aczel’s approach in the definition of our formalization of the set theory. We encode the sets within the

Nuprl type theory. There are two main reasons for formalizing *CZF*. First, we demonstrate the method for *deriving* one non-trivial theory from another (*CZF* is *derived* from the Nuprl type theory). In addition, set theory can often present a convenient framework for developing constructive mathematics using ordinary mathematical concepts. Aczel continues,

“Why not present constructive mathematics directly in the type theory? This is an obvious option for the constructive mathematician. It has the drawback that there is no extensive tradition of presenting mathematics in a type theoretic setting. So, many techniques for representing mathematical ideas in a set theoretical language have to be reconsidered for a type theoretical language. This can be avoided by keeping to the set theoretical language.”

The *CZF* set theory is defined in an intuitionistic first-order logic, with one additional primitive operator for set membership. All non-propositional elements of the set theory are sets; the numbers and other structures are coded in the usual manner. Sets use an *extensional* equality; two sets considered equal if they have the same elements.

## 4.1 Czf\_itt\_theory module

The *CZF* set theory is defined using a collection of axioms about set membership in a manner similar to the definition of *ZF* set theory. At the moment, the extent of our formalization is to encode the sets and prove that each of these axioms is valid given our interpretation of sets in the Nuprl type theory (defined in the `Itt_theory` module).

### 4.1.1 Axioms

#### Pairing

$$\forall a. \forall b. \exists c. \forall y. (y \in c \Leftrightarrow y = a \vee y = b)$$

#### Union

$$\forall a. \exists b. \forall y. (y \in b \Leftrightarrow \exists x \in a. (y \in x))$$

#### Infinity

$$\exists a. (\exists x. x \in a \wedge \forall x \in a. \exists y \in a. x \in y)$$

#### Extensionality

$$\forall a. \forall b. ((\forall x. x \in a \Leftrightarrow x \in b) \Rightarrow a = b)$$

### Set induction

$$\forall a.((\forall x \in a.\phi(x) \Rightarrow \phi(a)) \Rightarrow (\forall a.\phi(a)))$$

### Restricted separation

The constructive version of separation is stated for an arbitrary *restricted* formula  $\phi$ , where a formula is restricted is all quantifiers are restricted (either  $\exists x \in y. \dots$  or  $\forall x \in y. \dots$ ).

$$\forall a.\exists b.\forall x.(x \in b \Leftrightarrow (x \in a \wedge \phi(x)))$$

### Collection

The *CZF* set theory uses the *collection* schemes instead of the general replacement axiom, which would be defined as follows:

$$\forall a.((\forall x \in a.\exists!y.\phi(x, y)) \Rightarrow \exists b.\forall y.(y \in b \Leftrightarrow \exists x \in a.\phi(x, y)))$$

The collection axioms are defined in terms of a predicate  $\mathbb{B}$ .

$$\mathbb{B}x \in a, y \in b.\phi \equiv (\forall x \in a.\exists y \in b.\phi) \wedge (\forall y \in b.\exists x \in a.\phi)$$

The *strong collection* scheme is defined as follows:

$$\forall a.(\forall x \in a.\exists y.\phi(x, y)) \Rightarrow (\exists b.\mathbb{B}x \in a, y \in b.\phi(x, y)).$$

The *subset collection* scheme defines a collection of subsets.

$$\forall a.\forall b.\exists c.(\forall u.\forall x \in a.\exists y \in b.\phi(x, y, u) \Rightarrow \exists z \in c.\mathbb{B}x \in a, y \in b.\phi(x, y, u))$$

## 4.1.2 Theory organization

For our formalization of the set theory, we present a rule-based approach more suitable to a presentation in a sequent calculus. We present the set theory in three parts. First we introduce the primitive propositions for membership and equality; we then characterize the first order logic; and in the final section we present set *constructors* that correspond to each of the axioms of the set theory, together with well-formedness rules. The rules for the constructors are presented in the form of introduction and elimination rules for reasoning about membership in each of the set constructions. The following is a brief summary of the the account.

### Encoding

The type of sets is defined in `Czf_itt_set` module as the *W*-type  $set \equiv W(T: \mathbb{U}_1.T)$ . That is, each set has an index type  $T \in \mathbb{U}_1$ , and a function  $T \rightarrow set$  that lists the elements of the set.

## Propositions

**Equality**  $equal(s_1, s_2)$  The `Czf_itt_eq` module defines extensional set equality. The type-theoretic equality on the  $W$ -type is too fine for use in the set theory — two sets will differ if they have different index types, even though they may have the same elements. The `Czf_itt_set` module contains a discussion of the inadequacy of a quotient construction. Instead the `Czf_itt_eq` module defines several *functionality* judgments that are used to state well-formedness and functionality properties of set propositions and constructors.

**Membership**  $s_1 \in_s s_2$  The `Czf_itt_member` module defines membership, and proves the theorem of *extensionality*: two sets are equal if they have the same elements.

**Logic** The modules in `Czf_itt_fol` define the first-order logic, including the restricted quantifiers  $\forall x \in_s s. P[x]$  and  $\exists x \in_s s. P[x]$ , and the unrestricted quantifiers  $\forall_s x. P[x]$  and  $\exists_s x. P[x]$ . The modules in the logic prove the functionality of the logical operators in addition to the *restricted*( $P$ ) judgment for use in the separation construction.

**Subset**  $s_1 \subseteq_s s_2$  The subset judgment  $s_1 \subseteq_s s_2$  is defined in the `Czf_itt_subset` module as the proposition  $\forall x \in_s s_1. x \in_s s_2$ .

## Constructors

**Separation**  $\{x \in_s s \mid P[x]\}$  The `Czf_itt_sep` defines the constructor for restricted separation  $\{x \in_s s \mid P[x]\}$ , where  $s$  is a set, and  $P[x]$  must be a *restricted* proposition for any set  $x$ . The definition of restriction is equivalent to Aczel's definition (that  $P[x]$  contain no unrestricted quantifiers), but we give it a somewhat more general definition  $restricted(P) \equiv P \in \mathbb{U}_1$ . This definition is never exposed in the set theory; we provide rules for proving restriction for each of the restricted logical operators.

**Empty set**  $\{\}$  The empty set is defined in the `Czf_itt_empty` module.

**Singleton set**  $\{s\}$  The singleton  $\{s\}$  set, defined in the `Czf_itt_singleton` module, defines a set containing one element, the set  $s$ .

**Binary union**  $s_1 \cup s_2$ , **general union**  $\bigcup s$  The `Czf_itt_union` module defines two forms of union. The binary union  $s_1 \cup s_2$  is the set containing the elements of  $s_1$  and  $s_2$ . The general union  $\bigcup s$  forms the set containing the elements of all the sets in  $s$ .

**Pairing**  $(s_1, s_2)$  The `Czf_itt_pair` module defines the unordered pair  $(s_1, s_2)$ , which is a set containing the two elements  $s_1$  and  $s_1$ .

**Infinity (the natural numbers)**  $\omega$  The `Czf_itt_nat` module defines an infinite set  $\omega$  that corresponds to the set of natural numbers. It also defines a

zero element  $0_s \in_s \omega$ , and a successor  $s(i)$ , where  $s(i) \in_s \omega$  if  $i \in_s \omega$ . We also formalize an induction principle over the natural numbers, as well as the total order  $i <_s j$ .

### 4.1.3 Parents

**extends** Czf\_itt\_subset  
**extends** Czf\_itt\_power

## 4.2 Czf\_itt\_set module

The `Czf_itt_set` module provides the basic definition of sets and their elements. The `set` term denotes the type of all sets, defined using the  $W$ -type in the module `Itt_w`, as follows:

$$set \equiv W(T: \mathbb{U}_1.T).$$

That is, the *sets* are pairs of a type  $T \in \mathbb{U}_1$ , and a function  $T \rightarrow set$  that specifies the elements of the set. Note that the type  $T$  can be *any* type in  $\mathbb{U}_1$ ; equality of sets is in general undecidable, and their members can't necessarily be enumerated. This is a *constructive* theory, not a decidable theory. Of course, there will be special cases where equality of sets is decidable.

The sets are defined with the terms  $collect(x \in T.f[x])$ , where  $T$  is a type in  $\mathbb{U}_1$ , and  $f[x]$  is a set for any index  $x \in T$ . The sets  $f[x]$  are the *elements* of the set, and  $T$  is the a type used as their index. For example, the following set is empty.

$$\{\} = collect(x \in Void.x)$$

The following set is the singleton set containing the empty set.

$$\{\{\}\} = collect(x \in Unit.\{\})$$

The following set is equivalent.

$$\{\{\}\}' = collect(x \in \mathbb{Z}.\{\})$$

This raises an important point about equality. The membership equality defined on  $W$ -types requires equality on the index type  $T$  as well as the element function  $f$ . The two sets  $\{\{\}\}$  and  $\{\{\}\}'$  are *not* equal in this type because they have the provably different index types `Unit` and  `$\mathbb{Z}$` , even though they have the same elements.

One solution to this problem would be to use a quotient construction using the quotient type defined in the `Itt_quotient` module. The `Czf_itt_eq` module

defines extensional set equality  $\equiv_{ext}$ , and we could potentially define the “real” sets with the following type definition.

$$real\_sets \equiv (s_1, s_2: set // s_1 \equiv_{ext} s_2)$$

This type definition would require explicit functionality reasoning on all functions over *real.set*. This construction, however, makes these functionality proofs impossible because it omits the computational content of the equivalence judgment, which is a necessary part of the proof.

Alternative quotient formulations may be possible, but we have not pursued this direction extensively. Instead, we introduce set equality in the `Czf_itt_eq` module, together with explicit functionality predicates for set operators. In addition, we prove the functionality properties for all the primitive set operations.

One avenue for improvement in this theory would be to stratify the set types to arbitrary type universes  $\mathbb{U}_i$ , which would allow for higher-order reasoning on sets, classes, etc.

### 4.2.1 Parents

```
extends Itt_theory
extends Itt_eta
```

### 4.2.2 Terms

The `set` term defines the type of sets; the `collect` terms are the individual sets. The `isset` term is the well-formedness judgment for the *set* type. The `set_ind` term is the induction combinator for computation over sets. The *s* argument represents the set; *T* is its index type, *f* is its function value, and *g* is used to perform recursive computations on the children.

```
declare Czf_itt_set!set (displayed as set)
declare Czf_itt_set!isset{'s} (displayed as s set)
declare
  Czf_itt_set!collect{'T; x. a['x]}
  (displayed as collect x : T. a[x])
declare
  Czf_itt_set!set_ind{'s; T, f, g. b['T; 'f; 'g]}
  (displayed as let set(T, f).g = s in b[T; f; g])
```

### 4.2.3 Rewrites

The following four rewrites give the primitive definitions of the set constructions from the terms in the `Nuprl` type theory.

```

![] rewrite unfold_set :  $set \longleftrightarrow (\mathbb{W}x : \mathbb{U}_1.x)$ 
![] rewrite unfold_isset :  $(s \text{ set}) \longleftrightarrow (s \in \text{set})$ 
![] rewrite unfold_collect :
   $(\text{collect } x : T. a[x]) \longleftrightarrow \text{tree}(T, (\lambda x. a[x]))$ 
![] rewrite unfold_set_ind :
   $(\text{let } \text{set}(x, f).g = s \text{ in } b[x; f; g]) \longleftrightarrow$ 
     $(\text{tree\_ind}(g. \text{let } \text{tree}(x, f) = s \text{ in } b[x; f; g]))$ 

```

The `set_ind` term performs a pattern match; the normal reduction sequence can be derived from the computational behavior of the `tree_ind` term.

```

*[1, 147] rewrite reduce_set_ind { | reduce | } :
   $(\text{let } \text{set}(a, f).g = \text{collect } x : T. A[x] \text{ in } b[a; f; g]) \longleftrightarrow$ 
     $b[T;$ 
       $(\lambda x. A[x];$ 
         $(\lambda a_2. (\text{let } \text{set}(a, f).g = A[a_2] \text{ in } b[a; f; g]))]$ 

```

## 4.2.4 Rules

### Typelhood and equality

The `set` term is a type in the `Nuprl` type theory. The `equal_set` and `isset_assum` rules define the `isset` well-formedness judgment. The `isset_assum` is added to the `trivialT` tactic for use as default reasoning.

```

*[1, 50] rule set_type { | intro [] | } :
  [ext]  $\langle \Gamma \rangle \vdash \text{set Type}$ 
*[2, 24] rule equal_set :
  [ext]  $\langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash s \in \text{set}$ 
*[1, 8] rule isset_assum  $\Gamma$ :
  [ext]  $\langle \Gamma \rangle; x : \text{set}; \langle \Delta[x] \rangle \vdash x \text{ set}$ 

```

The `collect` terms are well-formed, if their index type  $T$  is a type in  $\mathbb{U}_1$ , and their element function  $a$  produces a set for any argument  $x \in T$ .

```

*[4, 99] rule isset_collect { | intro [] | } :
  [·]  $\langle \Gamma \rangle \vdash T \in \mathbb{U}_1 \longrightarrow$ 
  [·]  $\langle \Gamma \rangle; y : T \vdash a[y] \text{ set} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash \text{collect } x : T. a[x] \text{ set}$ 
*[1, 26] rule isset_collect2 { | intro [] | } :
  [·]  $\langle \Gamma \rangle \vdash T \in \mathbb{U}_1 \longrightarrow$ 
  [·]  $\langle \Gamma \rangle; y : T \vdash a[y] \text{ set} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash \text{collect } x : T. a[x] \in \text{set}$ 

```

## Elimination

The elimination form performs induction on the assumption  $a : \text{set}$ . The inductive argument is this: goal  $C$  is true for any set  $a$  if it is true for some set  $\text{collect}(x \in T.f(x))$  where the induction hypothesis is true on every child  $f(x)$  for  $x \in T$ . By definition, induction requires that tree representing the set be well-founded (which is true for all  $W$ -types).

```
*[10, 141] rule set_elim { | elim [(ThinOption thinT)] | }  $\Gamma$ :
  [ext]
  1.  $\langle \Gamma \rangle$ 
  2.  $a : \text{set}$ 
  3.  $\langle \Delta[a] \rangle$ 
  4.  $T : \mathbb{U}_1$ 
  5.  $f : T \rightarrow \text{set}$ 
  6.  $w : \forall x : T. C[(f\ x)]$ 
  7.  $z : \text{collect } x : T. f\ x\ \text{set}$ 
   $\vdash C[\text{collect } x : T. f\ x]$   $\longrightarrow$ 
  [ext]  $\langle \Gamma \rangle ; a : \text{set} ; \langle \Delta[a] \rangle \vdash C[a]$ 
```

## Combinator equality

The induction combinator computes a value of type  $T$  if its argument  $z$  is a set, and the body  $b[z, f, g]$  computes a value of type  $T$ , for any type  $z \in \mathbb{U}_1$ , any function  $f \in z \rightarrow \text{set}$ , and a recursive invocation  $g \in x : \mathbb{U}_1 \rightarrow x \rightarrow T$ .

```
*[5, 44] rule set_ind_equality2 { | intro [] | } :
  [wf] [·]  $\langle \Gamma \rangle \vdash z_1 = z_2 \in \text{set} \longrightarrow$ 
  [main] [·]
  1.  $\langle \Gamma \rangle$ 
  2.  $a_1 : \mathbb{U}_1$ 
  3.  $f_1 : a_1 \rightarrow \text{set}$ 
  4.  $g_1 : x : a_1 \rightarrow T$ 
   $\vdash \text{body}_1[a_1; f_1; g_1] = \text{body}_2[a_1; f_1; g_1] \in T \longrightarrow$ 
  [ext]
  1.  $\langle \Gamma \rangle$ 
   $\vdash$ 
   $(\text{let } \text{set}(a_1, f_1).g_1 = z_1 \text{ in } \text{body}_1[a_1; f_1; g_1])$ 
   $= (\text{let } \text{set}(a_2, f_2).g_2 = z_2 \text{ in } \text{body}_2[a_2; f_2; g_2]) \in T$ 
```

## 4.3 Czf\_itt\_eq module

The `Czf_itt_eq` module defines *extensional* equality of sets. Sets are equal if they have the same members, and in addition the equality must be *computable*.

The basic definition of equality is given with the `eq` term, which is defined as follows.

$$\begin{aligned} eq(\text{collect}(x_1 \in T_1.f_1[x_1]), \text{collect}(x_2 \in T_2.f_2[x_2])) &\equiv \\ \forall x_1: T_1. \exists x_2: T_2. eq(f_1[x_1], f_2[x_2]) & \\ \wedge \forall x_2: T_2. \exists x_1: T_1. eq(f_1[x_1], f_2[x_2]) & \end{aligned}$$

This recursive definition requires that for each element of one set, there must be an equal element in the other set. The proof of an equality is a pair of functions that, given an element of one set, produce the equal element in the other set. Note that there is significant computational content in this judgment.

The `eq` judgment is a predicate in  $\mathbb{U}_1$  for any two sets. The `equal`( $s_1, s_2$ ) judgment adds the additional requirement  $s_1 \in \text{set} \wedge s_2 \in \text{set}$  (which raises it to a predicate in  $\mathbb{U}_2$ ).

In addition to the equality judgments, the `Czf_itt_eq` module also defines *functionality* judgments. The `fun_set`( $s.f[s]$ ) requires that the function  $f$  compute equal set values for equal set arguments. The `fun_prop`( $s.P[s]$ ) requires that for any two equal sets  $s_1$  and  $s_2$ :  $P[s_1] \Rightarrow P[s_2]$ . The `dfun_prop` term is even more stringent. In the judgment `dfun_prop`( $A: A.B[x]$ ), the term  $A$  is a type, and  $B[x]$  is a type for any  $x \in A$ . The judgment requires that a proof  $B[u_2]$  be computable from *any* two proofs  $u_1: A$  and  $u_2: A$  and a proof  $B[u_1]$ :

$$dfun\_prop(A: A.B[x]) \equiv u_1: A \rightarrow B[u_1] \rightarrow u_2: A \rightarrow B[u_2].$$

The `dfun_prop` term is used for functionality judgments on dependent types.

### 4.3.1 Parents

`extends Czf_itt_set`

### 4.3.2 Terms

```

declare Czf_itt_eq!eq{ 's1; 's2 } (displayed as eq(s1; s2))
declare Czf_itt_eq!equal{ 's1; 's2 } (displayed as s1 =' s2)
declare
  Czf_itt_eq!fun_set{z. f['z]} (displayed as  $\forall z.f[z]$  fun_set)
declare
  Czf_itt_eq!fun_prop{z. P['z]} (displayed as  $\forall z.P[z]$  fun_prop)
declare
  Czf_itt_eq!dfun_prop{u. A['u]; x, y. B['x; 'y]}
  (displayed as  $\forall u: A[u]. x, y.B[x; y]$  fun_prop)

```

### 4.3.3 Rewrites

The `eq` judgment requires that the two sets have the same elements.

```

![] rewrite unfold_eq :
  eq(s1; s2) ←→
    (let set(T1, f1).g1 = s1 in
     let set(T2, f2).g2 = s2 in
      (∀y1 : T1. ∃y2 : T2. eq((f1 y1); (f2 y2)))
      ∧ (∀y2 : T2. ∃y1 : T1. eq((f1 y1); (f2 y2))))
*[1, 241] rewrite reduce_eq {| reduce |} :
  eq((collect x1 : T1. f1[x1]); (collect x2 : T2. f2[x2])) ←→
    ((∀y1 : T1. ∃y2 : T2. eq(f1[y1]; f2[y2]))
     ∧ (∀y2 : T2. ∃y1 : T1. eq(f1[y1]; f2[y2])))

```

The `unfold_equal` term requires that, in addition, the two arguments must be sets.

```

![] rewrite unfold_equal :
  (s1 =' s2) ←→ ((s1 set ∧ (s2 set)) ∧ eq(s1; s2))
*[1, 225] rewrite reduce_equal {| reduce |} :
  (collect x1 : T1. f1[x1] =' collect x2 : T2. f2[x2]) ←→
    ((collect x1 : T1. f1[x1] set
     ∧ (collect x2 : T2. f2[x2] set))
     ∧ (∀y1 : T1. ∃y2 : T2. eq(f1[y1]; f2[y2]))
     ∧ (∀y2 : T2. ∃y1 : T1. eq(f1[y1]; f2[y2])))

```

The following four rewrites define the functionality judgments.

```

![] rewrite unfold_fun_set :
  (∀z.f[z] fun_set) ←→
    (∀s1 : set. ∀s2 : set. ((s1 =' s2) → (f[s1] =' f[s2])))
![] rewrite unfold_fun_prop :
  (∀z.P[z] fun_prop) ←→
    (∀s1 : set. ∀s2 : set. ((s1 =' s2) → P[s1] → P[s2]))
![] rewrite unfold_dfun_prop :
  (∀u : A[u]. x, y.B[x; y] fun_prop) ←→
    (∀s1 : set
     ∀s2 : set
     ((s1 =' s2) → u1 : A[s1] → B[s1; u1] → u2 : A[s2] → B[s2; u2]))

```

### 4.3.4 Rules

#### Typehood and equality

The `eq` judgment is well-formed if both arguments are sets. The *equality* if the `eq` judgment requires the set arguments to be equal (in the native `set` type).

$*[9, 483]$  **rule** `eq_equality1`  $\{|$  `intro [] |}  $\}$  :  
 $[\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \in \mathbb{U}_1$   
 $*[1, 22]$  rule eq_type  $\{|$  intro [] |}  $\}$  :  
 $[\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \text{ Type}$   
 $*[11, 281]$  rule eq_equality2  $\{|$  intro [] |}  $\}$  :  
 $[\cdot] \langle \Gamma \rangle \vdash s_1 = s_3 \in \text{set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash s_2 = s_4 \in \text{set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) = eq(s_3; s_4) \in \mathbb{U}_1$`

The `eq` judgment also requires the arguments to be sets, but in addition, and equality judgment  $equal(s_1, s_2)$  *implies* that both  $s_1$  and  $s_2$  are types.

$*[1, 152]$  **rule** `equal_type`  $\{|$  `intro [] |}  $\}$  :  
 $[\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash (s_1 = s_2) \text{ Type}$   
 $*[3, 91]$  rule equal_intro  $\{|$  intro [] |}  $\}$  :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash s_1 = s_2$   
 $*[2, 30]$  rule equal_isset_left  $s_2$  :  
 $[ext] \langle \Gamma \rangle \vdash s_1 = s_2 \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash s_1 \text{ set}$   
 $*[2, 30]$  rule equal_isset_right  $s_1$  :  
 $[ext] \langle \Gamma \rangle \vdash s_1 = s_2 \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash s_2 \text{ set}$`

## Equality is an equivalence relation

The `eq` judgment is reflexive, symmetric, and transitive.

$*[5, 392]$  **rule** `eq_ref`  $\{|$  `intro [] |}  $\}$  :  
 $[\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_1)$   
 $*[14, 947]$  rule eq_sym :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_2; s_1) \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2)$   
 $*[19, 1726]$  rule eq_trans  $s_2$  :`

$$\begin{array}{l}
[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_3 \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_2; s_3) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_1; s_3)
\end{array}$$

### Functionality

The  $fun\_set(z.f[z])$  judgment implies that  $f[z]$  is a set for any set  $z$ .

$$\begin{array}{l}
*[6, 133] \text{ rule eq\_isset } \Gamma (\forall z.f[z] \text{ fun\_set}): \\
[ext] \langle \Gamma \rangle; z : \text{ set}; \langle \Delta[z] \rangle \vdash \forall z.f[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle; z : \text{ set}; \langle \Delta[z] \rangle \vdash f[z] \text{ set}
\end{array}$$

The  $eq$  and  $equal$  judgments are *functional* with respect to their set arguments.

$$\begin{array}{l}
*[29, 1032] \text{ rule eq\_fun } \{ | \text{ intro } [] | \} : \\
[ext] \langle \Gamma \rangle \vdash \forall z.f_1[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.f_2[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.eq(f_1[z]; f_2[z]) \text{ fun\_prop} \\
*[29, 951] \text{ rule equal\_fun } \{ | \text{ intro } [] | \} : \\
[ext] \langle \Gamma \rangle \vdash \forall z.f_1[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.f_2[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.(f_1[z] = f_2[z]) \text{ fun\_prop}
\end{array}$$

### Substitution

The following two rules define substitution. Set  $s_1$  can be replaced by set  $s_2$  in a context  $P[s_1]$  if  $s_1$  and  $s_2$  are equal, and the context  $P[x]$  is *functional* on set arguments.

$$\begin{array}{l}
*[7, 187] \text{ rule eq\_hyp\_subst } \Gamma s_1 s_2 \text{ bind}(v.P[v]): \\
[ext] \langle \Gamma \rangle; x : P[s_1]; \langle \Delta[x] \rangle \vdash s_1 = s_2 \longrightarrow \\
[ext] \langle \Gamma \rangle; x : P[s_1]; \langle \Delta[x] \rangle; z : P[s_2] \vdash C[x] \longrightarrow \\
[ext] \langle \Gamma \rangle; x : P[s_1]; \langle \Delta[x] \rangle \vdash \forall z.P[z] \text{ fun\_prop} \longrightarrow \\
[ext] \langle \Gamma \rangle; x : P[s_1]; \langle \Delta[x] \rangle \vdash C[x] \\
*[7, 235] \text{ rule eq\_concl\_subst } s_1 s_2 \text{ bind}(v.C[v]): \\
[ext] \langle \Gamma \rangle \vdash s_1 = s_2 \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash C[s_2] \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.C[z] \text{ fun\_prop} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash C[s_1]
\end{array}$$

## Typehood of the functionality judgments

The `fun_set` judgment requires that its argument be a family of sets, and the `fun_prop` judgment requires that its argument be a family of propositions.

```
*[2, 158] rule fun_set_type { | intro [] | } :
  [·] ⟨Γ⟩; z : set ⊢ f[z] set →
  [ext] ⟨Γ⟩ ⊢ (∀z.f[z] fun_set) Type
*[1, 126] rule fun_prop_type { | intro [] | } :
  [·] ⟨Γ⟩; z : set ⊢ f[z] Type →
  [ext] ⟨Γ⟩ ⊢ (∀z.f[z] fun_prop) Type
```

The trivial cases, where the functionality argument does not depend on the set argument, are functional. The identity function is also functional.

```
*[1, 143] rule fun_set { | intro [] | } :
  [·] ⟨Γ⟩ ⊢ u set →
  [ext] ⟨Γ⟩ ⊢ ∀z.u fun_set
*[1, 71] rule fun_ref { | intro [] | } :
  [ext] ⟨Γ⟩ ⊢ ∀z.z fun_set
*[1, 100] rule fun_prop { | intro [] | } :
  [·] ⟨Γ⟩ ⊢ P Type →
  [ext] ⟨Γ⟩ ⊢ ∀z.P fun_prop
```

## Substitution

`setSubstT` The `setSubstT` tactic *substitutes* one set for another. The usage is `setSubstT i eq(s1, s2)`, which replaces all occurrences of the term  $s_1$  with the term  $s_2$  in clause  $i$ .

### 4.3.5 Tactics

`eqSetRefT`, `eqSetSymT`, `eqSetTransT` The three `eqSet` tactics apply equivalence relation reasoning for the `eq` set judgment.

## 4.4 Czf\_itt\_member module

The `Czf_itt_member` module defines membership in a set. The basic definition is an existential judgment: a set  $s$  is an element of a set  $collect(x \in T.f[x])$  if there is some element  $a:T$  and  $eq(s, f[a])$ .

Note that equality has to be defined *before* membership. We also prove the *extensionality* judgment here; two sets are equal if they have the same members.

### 4.4.1 Parents

`extends Czf_itt_eq`

### 4.4.2 Terms

The member term defines the membership judgment.

```
declare Czf_itt_member!mem{'x; 'y} (displayed as  $x \in s y$ )  
declare Czf_itt_member!member{'x; 'y} (displayed as  $x \in S y$ )
```

### 4.4.3 Rewrites

The member judgment is defined using the `set_ind` induction combinator.

```
![] rewrite unfold_mem :  
  ( $x \in s y$ )  $\longleftrightarrow$  (let  $set(T, f).g = y$  in  $\exists t : T. eq(x; (f t))$ )  
*[1, 75] rewrite reduce_mem {| reduce |} :  
  ( $x \in s$  collect  $y : T. f[y]$ )  $\longleftrightarrow$  ( $\exists t : T. eq(x; f[t])$ )  
![] rewrite unfold_member : ( $x \in S y$ )  $\longleftrightarrow$  ( $(x$  set  $\wedge$  ( $y$  set))  $\wedge$  ( $x \in s y$ ))  
*[1, 123] rewrite reduce_member {| reduce |} :  
  ( $x \in S$  collect  $y : T. f[y]$ )  $\longleftrightarrow$   
  ( $(x$  set  $\wedge$  (collect  $y : T. f[y]$  set))  $\wedge$  ( $\exists t : T. eq(x; f[t])$ ))
```

### 4.4.4 Rules

#### Well-formedness

The member judgment is well-formed if-and-only-if its arguments are sets.

```
*[6, 224] rule mem_type {| intro [] |} :  
  [wf] [.]  $\langle \Gamma \rangle \vdash s_1$  set  $\longrightarrow$   
  [wf] [.]  $\langle \Gamma \rangle \vdash s_2$  set  $\longrightarrow$   
  [ext]  $\langle \Gamma \rangle \vdash (s_1 \in s s_2)$  Type  
*[7, 241] rule mem_equal {| intro [] |} :  
  [wf] [.]  $\langle \Gamma \rangle \vdash s_1$  set  $\longrightarrow$   
  [wf] [.]  $\langle \Gamma \rangle \vdash s_2$  set  $\longrightarrow$   
  [ext]  $\langle \Gamma \rangle \vdash s_1 \in s s_2 \in \mathbb{U}_1$   
*[3, 91] rule member_intro {| intro [] |} :  
  [wf] [.]  $\langle \Gamma \rangle \vdash s_1$  set  $\longrightarrow$   
  [wf] [.]  $\langle \Gamma \rangle \vdash s_2$  set  $\longrightarrow$   
  [ext]  $\langle \Gamma \rangle \vdash s_1 \in s s_2 \longrightarrow$   
  [ext]  $\langle \Gamma \rangle \vdash s_1 \in S s_2$   
*[2, 34] rule elem_isset  $y$  :  
  [ext]  $\langle \Gamma \rangle \vdash x \in S y \longrightarrow$ 
```

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash x \text{ set} \\
*[2, 34] \text{ rule set\_isset } x: \\
[ext] \langle \Gamma \rangle \vdash x \in S y \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash y \text{ set}
\end{array}$$

### Functionality

The `member` judgment is functional in both its arguments. The next two rules provide simple functionality judgments for the two set arguments.

$$\begin{array}{l}
*[10, 505] \text{ rule mem\_fun\_left } s_1: \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_3 \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s_1 \in s s_3 \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s_2 \in s s_3
\end{array}$$

nil

$$\begin{array}{l}
*[12, 860] \text{ rule mem\_fun\_right } s_1: \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_3 \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_1; s_2) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s_3 \in s s_1 \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s_3 \in s s_2
\end{array}$$

The `member_fun` rule proves the general functionality judgment.

$$\begin{array}{l}
*[20, 514] \text{ rule member\_fun } \{ | \text{ intro } [] \} : \\
[ext] \langle \Gamma \rangle \vdash \forall z. f_1[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z. f_2[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z. f_1[z] \in s f_2[z] \text{ fun\_prop}
\end{array}$$

### Set extensionality

Two sets are equal if-and-only-if they have the same elements. The proof of this theorem is straightforward. The two membership goals are the functions that “choose,” for any element of one set, an equal element in the other set. The equality judgment can be proved with the pair of both choice functions.

$$\begin{array}{l}
*[20, 1256] \text{ rule set\_ext} : \\
[wf] [ext] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[wf] [ext] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow
\end{array}$$

$$\begin{array}{l}
[main] \ [ext] \ \langle \Gamma \rangle; x : set; y : x \in s \ s_1 \vdash x \in s \ s_2 \longrightarrow \\
[main] \ [ext] \ \langle \Gamma \rangle; x : set; y : x \in s \ s_2 \vdash x \in s \ s_1 \longrightarrow \\
[ext] \ \langle \Gamma \rangle \vdash eq(s_1; s_2)
\end{array}$$

#### 4.4.5 Tactics

`memberOfT`, `setOfT` The `memberOfT` applies the `elem_isset` rule, and the `setOfT` tactic applies the `set_isset` rule. Both tactics infer the well-formedness of a set for a membership or equality judgment.

`setExtT` The `setExtT` tactic refines a set-equality goal using the rule of *extensionality* `set_ext`. This rule is not added to the `dT` tactic for default reasoning because in many cases, equality judgments can be proved in simpler ways.

### 4.5 Czf\_itt\_fol module

The `Czf_itt_fol` module defines the first-order logic for the set theory. The predicates are defined as follows.

$$\begin{array}{ll}
false_s & \equiv \ Void \\
true_s & \equiv \ Unit \\
A \vee_s B & \equiv \ A + B \\
A \wedge_s B & \equiv \ A \times B \\
A \Rightarrow_s B & \equiv \ A \rightarrow B \\
\exists_s x: A.B[x] & \equiv \ x: A \times B[x] \\
\forall_s x: A.B[x] & \equiv \ x: A \rightarrow B[x]
\end{array}$$

The first-order logic is defined in seven modules, one for each logical operator: `Czf_itt_false`, `Czf_itt_true`, `Czf_itt_or`, `Czf_itt_and`, `Czf_itt_implies`, `Czf_itt_exists`, and `Czf_itt_all`.

We omit the modules, the proofs are straightforward. The goal of these modules is to prove *functionality* and *restriction* of each of the operators. The restriction judgment follows directly from the well-formedness of the logical operators. The disjunction demonstrates a typical argument of functionality for a propositional operator.

**Functionality of disjunction** The goal is `fun_prop(z.A[z]  $\vee_s$  B[z])`, assuming `fun_prop(z.A[z])` and `fun_prop(z.B[z])`. To prove this, assume `A[s1]  $\vee_s$  B[s1]` for some set `s1`, and prove `A[s2]  $\vee_s$  B[s2]` for some set `s2 = s1`.

First, perform a case analysis on `A[s1]  $\vee_s$  B[s1]`. In the case where `A[s1]`, a proof of `A[s2]` can be constructed from the proof of functionality of `A`. This proof is sufficient for proving `A[s2]  $\vee_s$  B[s2]`. The case for `B` is similar.

The quantifiers are somewhat more difficult. Strictly speaking, the quantifiers are unnecessary, because the set theory can't reason about the programs in the first-order logic directly. The usual set-theoretic quantifiers are defined in the `Czf_itt_dall`, `Czf_itt_dexists`, `Czf_itt_sall`, and `Czf_itt_sexists` modules. The existential demonstrates a typical functionality argument.

**Functionality of existential quantification** The goal is to prove  $fun\_prop(z.w: A[z] \times B[z, w])$ , given  $fun\_prop(z.A[z])$  and  $dfun\_prop(A[z]: A[z].B[z, w])$  for any set  $z$ . This means that we must produce a proof of  $w: A[z_1] \times B[z_1, w]$  from a proof of  $w: A[z_2] \times B[z_2, w]$  for any two equal sets  $z_1$  and  $z_2$ .

The assumption  $fun\_prop(z.A[z])$  means that a proof of  $A[z_2]$  can be computed from a proof  $A[z_1]$  for any equal sets  $z_1$  and  $z_2$ . The second assumption means that a proof  $B[z_2, w_2]$  can be found given *any* proofs  $w_1: A[z_1]$ ,  $w_2: A[z_2]$ , and  $B[z_1, w_1]$ . The proof of the goal is constructed from the proof  $w_2: A[z_2]$  and the proof of  $B[z_2, w_2]$ .

## 4.6 Czf\_itt\_dall module

The `Czf_itt_dall` theory defines *restricted* universal quantification. The syntax of the operator is  $\forall x \in_s s.P[x]$ , where  $s$  is a set, and  $P[x]$  is a functional proposition for any set argument  $x$ . The proposition is true if  $P[a]$  is true for any element  $a \in_s s$ .

The restricted universal quantification is coded as an implication on the elements of  $s$ .

$$\forall x \in_s collect(y \in T.f[y]).P[x] \equiv \forall x: T.P[f(x)]$$

### 4.6.1 Parents

```
extends Czf_itt_all
extends Czf_itt_set.ind
```

### 4.6.2 Terms

```
declare Czf_itt_dall!dall{'T; x. A['x]} (displayed as  $\forall x \in_s T.A[x]$ )
```

### 4.6.3 Rewrites

The `dall` term is defined by set induction on the set argument as a universal quantification over the index type.

![] **rewrite** `unfold_dall` :  
 $(\forall x \in s s.A[x]) \longleftrightarrow (\text{let } \text{set}(a, f).g = s \text{ in } x: a \rightarrow A[(f\ x)])$   
 \*[1, 65] **rewrite** `reduce_dall` { | `reduce` | } :  
 $(\forall y \in s \text{ collect } x : T. f[x].A[y]) \longleftrightarrow t: T \rightarrow A[f[t]]$

#### 4.6.4 Rules

##### Well-formedness

The  $\forall x \in_s s.P[x]$  proposition is well-formed if  $s$  is a set, and  $P[x]$  is a proposition for any set  $x$ .

\*[5, 245] **rule** `dall_type` { | `intro` [] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; y: \text{set} \vdash A[y] \text{Type} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash (\forall x \in s s.A[x]) \text{Type}$

##### Introduction

The proposition  $\forall x \in_s s.P[x]$  is true if it is well-formed, and  $P[a]$  is true for any set  $a \in_s s$ .

\*[14, 864] **rule** `dall_intro` { | `intro` [] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; a: \text{set} \vdash A[a] \text{Type} \longrightarrow$   
 $[main] [ext] \langle \Gamma \rangle; a: \text{set}; b: a \in_s s \vdash A[a] \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \forall x \in_s s.A[x]$

##### Elimination

The elimination form instantiates the universal quantification  $\forall x \in_s s.P[x]$  on a particular argument  $z \in_s s$ . It produces an additional assumption  $A[z]$ .

\*[21, 681] **rule** `dall_elim` { | `elim` [] | }  $\Gamma z$  :  
 $[wf] [\cdot] \langle \Gamma \rangle; x: \forall y \in_s s.A[y]; \langle \Delta[x] \rangle; w: \text{set} \vdash A[w] \text{Type} \longrightarrow$   
 $[wf] [ext] \langle \Gamma \rangle; x: \forall y \in_s s.A[y]; \langle \Delta[x] \rangle \vdash \forall w.A[w] \text{fun\_prop} \longrightarrow$   
 $[antecedent] [ext] \langle \Gamma \rangle; x: \forall y \in_s s.A[y]; \langle \Delta[x] \rangle \vdash z \in_s s \longrightarrow$   
 $[main] [ext] \langle \Gamma \rangle; x: \forall y \in_s s.A[y]; \langle \Delta[x] \rangle; w: A[z] \vdash C[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x: \forall y \in_s s.A[y]; \langle \Delta[x] \rangle \vdash C[x]$

##### Functionality

The `dall` proposition is functional in its set and proposition arguments.

\*[27, 1597] **rule** `dall_fun`  $\{ | \text{intro } [] | \}$  :

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash \forall z. A[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle; z : \text{set} \vdash \forall x. B[z; x] \text{ fun\_prop} \longrightarrow \\
[ext] \langle \Gamma \rangle; z : \text{set} \vdash \forall x. B[x; z] \text{ fun\_prop} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle; z : \text{set}; x : \text{set} \vdash B[z; x] \text{Type} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z. (\forall y \in s A[z]. B[z; y]) \text{ fun\_prop}
\end{array}$$

### Restriction

The proposition  $\forall x \in_s s. P[x]$  is restricted for any set  $s$  because the proposition quantifies over the *index* type of the set argument (which is a restricted proposition itself).

\*[6, 149] **rule** `dall_res`  $\{ | \text{intro } [] | \}$  :

$$\begin{array}{l}
[wf] [\cdot] \langle \Gamma \rangle \vdash A \text{ set} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; x : \text{set} \vdash \langle B[x] \rangle \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \langle (\forall y \in s A.B[y]) \rangle
\end{array}$$

## 4.7 Czf\_itt\_dexists module

The `Czf_itt_dexists` theory defines *restricted* existential quantification. The syntax of the operator is  $\exists x \in_s s. P[x]$ , where  $s$  is a set, and  $P[x]$  is a functional proposition for any set argument  $x$ . The proposition is true if  $P[a]$  is true for *some* element  $a \in_s s$ .

The restricted universal quantification is coded as an implication on the elements of  $s$ .

$$\exists x \in_s \text{ collect}(y \in T. f[y]). P[x] \equiv x : T \times P[x]$$

### 4.7.1 Parents

`extends` `Czf_itt_sep`  
`extends` `Czf_itt_set_ind`

### 4.7.2 Terms

`declare`  
`Czf_itt_dexists!dexists`  $\{ 'T; x. A[ 'x] \}$   
`(displayed as`  $\exists x \in_s T. A[x]$ `)`

### 4.7.3 Rewrites

The existential is defined by set induction on the set argument as a dependent product type.

```
![] rewrite unfold_dexists :
  (∃x ∈ s s.A[x]) ↔ (let set(T, f).g = s in (x : T × A[f x]))
*[1, 65] rewrite reduce_dexists {[] reduce []} :
  (∃y ∈ s collect x : T. f[x].A[y]) ↔ (t : T × A[f[t]])
```

### 4.7.4 Rules

#### Well-formedness

The proposition  $\exists x \in_s s.P[x]$  is well-formed if  $s$  is a set, and  $P[x]$  is a well-formed proposition for *any* set argument  $x$ .

```
*[5, 245] rule dexists_type {[] intro [] []} :
  [wf] [·] ⟨Γ⟩ ⊢ s set →
  [wf] [·] ⟨Γ⟩; y : set ⊢ A[y] Type →
  [ext] ⟨Γ⟩ ⊢ (∃x ∈ s s.A[x]) Type
```

#### Introduction

The existential  $\exists x \in_s s.P[x]$  is true if it is well-formed and if  $P[a]$  is true for some element  $a \in_s s$ .

```
*[13, 848] rule dexists_intro {[] intro [] []} z :
  [wf] [·] ⟨Γ⟩; w : set ⊢ A[w] Type →
  [wf] [ext] ⟨Γ⟩ ⊢ ∀x.A[x] fun_prop →
  [main] [ext] ⟨Γ⟩ ⊢ z ∈ S s →
  [antecedent] [ext] ⟨Γ⟩ ⊢ A[z] →
  [ext] ⟨Γ⟩ ⊢ ∃x ∈ s s.A[x]
```

#### Elimination

The proof of the existential  $\exists x \in_s s.P[x]$  has two parts: an element  $a \in_s s$ , and a proof  $P[a]$ . The elimination form produces these parts.

```
*[11, 650] rule dexists_elim {[] elim [] []} Γ :
  [wf] [·] ⟨Γ⟩; x : ∃y ∈ s s.A[y]; ⟨Δ[x]⟩ ⊢ s set →
  [wf] [·] ⟨Γ⟩; x : ∃y ∈ s s.A[y]; ⟨Δ[x]⟩; z : set ⊢ A[z] Type →
  [wf] [ext] ⟨Γ⟩; x : ∃y ∈ s s.A[y]; ⟨Δ[x]⟩ ⊢ ∀z.A[z] fun_prop →
  [main] [ext]
  1. ⟨Γ⟩
  2. x : ∃y ∈ s s.A[y]
```

3.  $\langle \Delta[x] \rangle$
4.  $z : \text{set}$
5.  $v : z \in s \ s$
6.  $w : A[z]$

$$\vdash C[x] \longrightarrow$$

$$[\text{ext}] \langle \Gamma \rangle; x : \exists y \in s \ s.A[y]; \langle \Delta[x] \rangle \vdash C[x]$$

### Functionality

The existential is functional in both its set and proposition arguments.

\*[13, 551] **rule dexists\_fun** { | intro [] | } :

$$[\text{ext}] \langle \Gamma \rangle \vdash \forall z.A[z] \text{ fun\_set} \longrightarrow$$

$$[\text{ext}] \langle \Gamma \rangle; z : \text{set} \vdash \forall x.B[z; x] \text{ fun\_prop} \longrightarrow$$

$$[\text{ext}] \langle \Gamma \rangle; z : \text{set} \vdash \forall x.B[x; z] \text{ fun\_prop} \longrightarrow$$

$$[\text{wf}] [\cdot] \langle \Gamma \rangle; z : \text{set}; x : \text{set} \vdash B[z; x] \text{Type} \longrightarrow$$

$$[\text{ext}] \langle \Gamma \rangle \vdash \forall z.(\exists y \in s \ A[z].B[z; y]) \text{ fun\_prop}$$

### Restriction

The existential is a restricted formula because it is a quantification over the *index* type of the set argument.

\*[5, 147] **rule dexists\_res2** { | intro [] | } :

$$[\text{wf}] [\cdot] \langle \Gamma \rangle \vdash A \text{ set} \longrightarrow$$

$$[\cdot] \langle \Gamma \rangle; u : \text{set} \vdash \langle B[u] \rangle \longrightarrow$$

$$[\text{ext}] \langle \Gamma \rangle \vdash \langle \exists y \in s \ A.B[y] \rangle$$

## 4.8 Czf\_itt\_sall module

The `Czf_itt_sall` module defines the *unrestricted* universal quantification  $\forall_s x.P[x]$  over all sets  $x$ . The proposition  $P[x]$  must be well-formed for all sets  $x$ .

### 4.8.1 Parents

`extends Czf_itt_set`

### 4.8.2 Terms

`declare Czf_itt_sall!sall{x. A['x]}` (displayed as  $\forall_s x.A[x]$ )

### 4.8.3 Rewrites

The quantification  $\forall_s x.P[x]$  is defined using the universal quantifier `all` from the `Itt_logic` module.

```
![] rewrite unfold_sall : ( $\forall_s x.A[x]$ )  $\longleftrightarrow$  ( $\forall x : set. A[x]$ )
```

### 4.8.4 Rules

#### Well-formedness

The quantification  $\forall_s x.A[x]$  is well-formed if  $A[x]$  is a proposition for any set  $x$ .

```
*[1, 56] rule sall_type { | intro [] | } :
  [ext]  $\langle \Gamma \rangle ; y : set \vdash A[y] \text{Type} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (\forall_s x.A[x]) \text{Type}$ 
```

#### Introduction

The quantification  $\forall_s x.A[x]$  is true if it is well-formed, and if  $A[x]$  is true for every set  $x$ .

```
*[1, 25] rule sall_intro { | intro [] | } :
  [ext]  $\langle \Gamma \rangle ; a : set \vdash A[a] \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash \forall_s x.A[x]$ 
```

#### Elimination

The elimination form instantiates the universal assumption on a particular set argument  $z$ .

```
*[3, 43] rule sall_elim { | elim [] | }  $\Gamma z :$ 
  [wf] [·]  $\langle \Gamma \rangle ; x : \forall_s y.A[y] ; \langle \Delta[x] \rangle \vdash z \text{set} \longrightarrow$ 
  [main] [ext]  $\langle \Gamma \rangle ; x : \forall_s y.A[y] ; \langle \Delta[x] \rangle ; w : A[z] \vdash T[x] \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle ; x : \forall_s y.A[y] ; \langle \Delta[x] \rangle \vdash T[x]$ 
```

## 4.9 Czf\_itt\_sexists module

The `Czf_itt_sexists` module defines the *unrestricted* existential quantification  $\exists_s x.P[x]$ . The proposition  $P[x]$  must be well-formed for any set argument. The existential is true, if  $P[a]$  is true for some set  $a$ .

### 4.9.1 Parents

`extends Czf_itt_and`

### 4.9.2 Terms

`declare Czf_itt_sexists!sexists{x. A['x]}` (displayed as  $\exists_s x.A[x]$ )

### 4.9.3 Rewrites

The unrestricted existential is defined with the type-theoretic existential `exists` from the `Itt_logic` module.

`![] rewrite unfold_sexists :  $(\exists_s x.A[x]) \longleftrightarrow (\exists x : set. A[x])$`

### 4.9.4 Rules

#### Well-formedness

The unrestricted existential  $\exists_s x.P[x]$  is well-formed if  $P[x]$  is a well-formed proposition for any set argument  $x$ .

`*[1, 49] rule sexists_type { | intro [] | } :`  
`[·]  $\langle \Gamma \rangle ; y : set \vdash A[y] \text{Type} \longrightarrow$`   
`[ext]  $\langle \Gamma \rangle \vdash (\exists_s x.A[x]) \text{Type}$`

#### Introduction

The existential  $\exists_s x.P[x]$  is true if  $P[a]$  is true for some set  $a$ .

`*[3, 48] rule sexists_intro { | intro [] | } z :`  
`[wf] [ext]  $\langle \Gamma \rangle \vdash z \text{ set} \longrightarrow$`   
`[main] [ext]  $\langle \Gamma \rangle \vdash A[z] \longrightarrow$`   
`[wf] [ext]  $\langle \Gamma \rangle ; w : set \vdash A[w] \text{Type} \longrightarrow$`   
`[ext]  $\langle \Gamma \rangle \vdash \exists_s x.A[x]$`

#### Elimination

The proof of the existential  $\exists_s x.P[x]$  is a pair of a witness set  $a$  and a proof  $P[a]$ .

`*[1, 10] rule sexists_elim { | elim [] | }  $\Gamma :$`   
`[ext]  $\langle \Gamma \rangle ; z : set ; w : A[z] ; \langle \Delta[(z, w)] \rangle \vdash T[(z, w)] \longrightarrow$`   
`[ext]  $\langle \Gamma \rangle ; x : \exists_s y.A[y] ; \langle \Delta[x] \rangle \vdash T[x]$`

## 4.10 Czf\_itt\_subset module

The `Czf_itt_subset` module defines the subset proposition  $\backslash subsets_{s_1; s_2}$ , which is a proposition for any two sets  $s_1$  and  $s_2$ . The subset is a derived proposition defined as follows.

$$\backslash subsets_{s_1; s_2} \equiv \forall x \in_s s_1. x \in_s s_2$$

### 4.10.1 Parents

`extends Czf_itt_dall`

### 4.10.2 Terms

`declare Czf_itt_subset!subset{ 's1; 's2} (displayed as  $s_1 \subseteq s s_2$ )`

### 4.10.3 Rewrites

The subset is defined using restricted universal quantification.

`![] rewrite unfold_subset : ( $s_1 \subseteq s s_2$ )  $\longleftrightarrow$  ( $\forall x \in_s s_1. x \in_s s_2$ )`

### 4.10.4 Rules

#### Well-formedness

The subset proposition  $\backslash subsets_{s_1; s_2}$  is well-formed if  $s_1$  and  $s_2$  are both sets.

```
*[1, 69] rule subset_type { | intro [] | } :
  [wf] [·] ⟨Γ⟩ ⊢ s1 set →
  [wf] [·] ⟨Γ⟩ ⊢ s2 set →
  [ext] ⟨Γ⟩ ⊢ (s1 ⊆ s s2) Type
```

#### Introduction

The subset proposition  $\backslash subsets_{s_1; s_2}$  is true if every element  $x \in_s s_1$  is also an element of  $s_2$ .

```
*[1, 50] rule subset_intro { | intro [] | } :
  [wf] [·] ⟨Γ⟩ ⊢ s1 set →
  [wf] [·] ⟨Γ⟩ ⊢ s2 set →
  [main] [ext] ⟨Γ⟩; x : set; y : x ∈ s s1 ⊢ x ∈ s s2 →
  [ext] ⟨Γ⟩ ⊢ s1 ⊆ s s2
```

## Elimination

The elimination form of the proposition  $\backslash subsets_1; s_2$  takes an element  $x \in_s s_1$  and it produces a proof that  $x \in_s s_2$ .

```
*[3, 153] rule subset_elim {| elim [] |} Γ s :
  [wf] [·] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩ ⊢ s set →
  [wf] [·] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩ ⊢ s1 set →
  [wf] [·] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩ ⊢ s2 set →
  [antecedent] [ext] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩ ⊢ s ∈ s s1 →
  [main] [ext] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩; z : s ∈ s s2 ⊢ C[x] →
  [ext] ⟨Γ⟩; x : s1 ⊆ s s2; ⟨Δ[x]⟩ ⊢ C[x]
```

## Functionality

The subset proposition is functional in both set arguments, and it is a restricted proposition.

```
*[1, 45] rule subset_res {| intro [] |} :
  [wf] [·] ⟨Γ⟩ ⊢ s1 set →
  [wf] [·] ⟨Γ⟩ ⊢ s2 set →
  [ext] ⟨Γ⟩ ⊢ < (s1 ⊆ s s2) >
*[3, 137] rule subset_fun {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ ∀z.s1[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.s2[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(s1[z] ⊆ s s2[z]) fun_prop
```

## 4.11 Czf\_itt\_sep module

The `Czf_itt_sep` module defines *restricted separation*. Separation is defined as a set constructor  $\{x \in_s s \mid P[x]\}$ . The term  $s$  must be a set, and  $P[x]$  must be a functional, restricted proposition. The elements of the separation are the elements of  $x \in s$  for which  $P[x]$  is true.

The separation constructor is a set if  $s$  is a set, and if  $P[x]$  is *restricted*. Our formalization of restriction differs slightly from Aczel's account, although both are equivalent. In Aczel's definition, a proposition  $P[x]$  is restricted if it contains no *unbounded* quantifiers  $\forall s \dots$  and  $\exists s \dots$ . In our construction, a predicate  $P$  is restricted if it is a well-formed proposition in  $\mathbb{U}_1$ .

The separation constructor  $\{z \in_s collect(x \in T.f[x]) \mid P[z]\}$  is defined with the product type as the set  $collect(z \in (x : A \times P[x]).f(fst(z)))$ .

### 4.11.1 Parents

`extends Czf_itt_member`

### 4.11.2 Terms

`declare`

`Czf_itt_sep!sep{'s; x. P['x]}` (displayed as  $\{x \in s \mid P[x]\}$ )  
`declare Czf_itt_sep!restricted{'P}` (displayed as  $\langle P \rangle$ )

### 4.11.3 Rewrites

The `sep` term is defined by set induction.

`![] rewrite unfold_sep :`  
 $\{x \in s \mid P[x]\} \longleftrightarrow$   
*(let set(T, f).g = s in collect z : (t : T × P[(f t])). f z.1)*  
`*[1, 101] rewrite reduce_sep { | reduce | } :`  
 $\{z \in s \text{ (collect } x : T. f[x] \mid P[z])\} \longleftrightarrow$   
*(collect w : (t : T × P[f[t]]). f[w.1])*

The `restricted(P)` predicate means that the proposition is well-formed in  $\mathbb{U}_1$ .

`![] rewrite unfold_restricted : ( $\langle P \rangle$ )  $\longleftrightarrow$  ( $P \in \mathbb{U}_1$ )`

### Equality and membership are restricted judgments

The next two rules show that equality and membership are restricted for any `set` arguments.

`*[8, 376] rule eq_restricted { | intro [] | } :`  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \langle eq(s_1; s_2) \rangle$   
`*[1, 42] rule member_restricted { | intro [] | } :`  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \langle s_1 \in s_2 \rangle$

### Well-formedness

The separation  $\{x \in_s s \mid P[x]\}$  is well-formed if  $s$  is a set, and  $P[x]$  is restricted and functional on any set argument  $x$ .

\*[7, 287] **rule sep\_isset**  $\{| \text{intro } [] \}$  :

$$\begin{array}{l} [wf] [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow \\ [wf] [ext] \langle \Gamma \rangle \vdash \forall z. P[z] \text{ fun\_prop} \longrightarrow \\ [wf] [\cdot] \langle \Gamma \rangle; z : \text{set} \vdash \langle P[z] \rangle \longrightarrow \\ [ext] \langle \Gamma \rangle \vdash \{ x \in s \mid P[x] \} \text{ set} \end{array}$$

## Introduction

A set  $x$  is a member of  $\{z \in s \mid P[z]\}$  if the separation is well-formed; if  $x \in_S s$ ; and  $P[x]$ .

\*[19, 1245] **rule sep\_intro2**  $\{| \text{intro } [] \}$  :

$$\begin{array}{l} [wf] [\cdot] \langle \Gamma \rangle; w : \text{set} \vdash \langle P[w] \rangle \longrightarrow \\ [wf] [ext] \langle \Gamma \rangle \vdash \forall z. P[z] \text{ fun\_prop} \longrightarrow \\ [main] [ext] \langle \Gamma \rangle \vdash x \in_S s \longrightarrow \\ [main] [ext] \langle \Gamma \rangle \vdash P[x] \longrightarrow \\ [ext] \langle \Gamma \rangle \vdash x \in s \{ z \in s \mid P[z] \} \end{array}$$

## Elimination

An assumption  $x \in_s \{y \in_s s \mid P[y]\}$  implies two facts:  $x \in_s s$  and  $P[x]$ . The computational content of the predicate  $P[x]$  is visible (unlike the separation “set” constructor in the Nuprl type theory module `Itt_set`).

\*[21, 843] **rule sep\_elim**  $\{| \text{elim } [] \}$   $\Gamma$ :

$$\begin{array}{l} [wf] [\cdot] \langle \Gamma \rangle; w : x \in s \{ y \in s \mid P[y] \}; \langle \Delta[w] \rangle \vdash x \text{ set} \longrightarrow \\ [wf] [\cdot] \langle \Gamma \rangle; w : x \in s \{ y \in s \mid P[y] \}; \langle \Delta[w] \rangle \vdash s \text{ set} \longrightarrow \\ [wf] [\cdot] \langle \Gamma \rangle; w : x \in s \{ y \in s \mid P[y] \}; \langle \Delta[w] \rangle; z : \text{set} \vdash \langle P[z] \rangle \longrightarrow \\ [wf] [ext] \langle \Gamma \rangle; w : x \in s \{ y \in s \mid P[y] \}; \langle \Delta[w] \rangle \vdash \forall z. P[z] \text{ fun\_prop} \longrightarrow \\ [main] [ext] \\ \quad 1. \langle \Gamma \rangle \\ \quad 2. w : x \in s \{ y \in s \mid P[y] \} \\ \quad 3. \langle \Delta[w] \rangle \\ \quad 4. u : x \in s \\ \quad 5. v : P[x] \\ \quad \vdash T[w] \longrightarrow \\ [ext] \langle \Gamma \rangle; w : x \in s \{ y \in s \mid P[y] \}; \langle \Delta[w] \rangle \vdash T[w] \end{array}$$

## Functionality

The separation constructor is functional in both the set argument and the proposition.

\*[16, 2516] **rule sep\_fun**  $\{| \text{intro } [] \}$  :

$$[\cdot] \langle \Gamma \rangle; u : \text{set}; v : \text{set} \vdash \langle P[u; v] \rangle \longrightarrow$$

$$\begin{array}{l}
[ext] \langle \Gamma \rangle; u : set \vdash \forall z. P[z; u] \text{ fun\_prop} \longrightarrow \\
[ext] \langle \Gamma \rangle; u : set \vdash \forall z. P[u; z] \text{ fun\_prop} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z. s[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z. \{ x \in s \ s[z] \mid P[x; z] \} \text{ fun\_set}
\end{array}$$

## 4.12 Czf\_itt\_empty module

The `Czf_itt_empty` module defines an empty set as the set  $collect(x \in Void.x)$ . Since the `Void` type is empty, the set has no elements.

### 4.12.1 Parents

`extends Czf_itt_member`

### 4.12.2 Terms

`declare Czf_itt_empty!empty (displayed as {})`

### 4.12.3 Rewrites

The empty set uses the empty index type `Void`.

`![] rewrite unfold_empty : {}  $\longleftrightarrow$  (collect x : Void. x)`

### 4.12.4 Rules

There are only two rules for the empty set: it is a well-formed set, and it has no elements.

$$\begin{array}{l}
*[1, 26] \text{ rule empty\_isset } \{ | \text{intro } [] \} : \\
\quad [ext] \langle \Gamma \rangle \vdash \{ \} \text{ set} \\
*[1, 27] \text{ rule empty\_member\_elim } \{ | \text{elim } [] \} \Gamma : \\
\quad [ext] \langle \Gamma \rangle; x : y \in s \{ \}; \langle \Delta[x] \rangle \vdash T[x]
\end{array}$$

## 4.13 Czf\_itt\_singleton module

The `Czf_itt_singleton` module defines the singleton sets  $\{s\}$ , which is a set that contains the single element  $s$ . The singleton is used as a building block for pairing, defined in the `Czf_itt_pair` module.

### 4.13.1 Parents

extends Czf\_itt\_member

### 4.13.2 Terms

declare Czf\_itt\_singleton!sing{'s} (displayed as {s})

### 4.13.3 Rewrites

The singleton set {s} is defined as a set with an index type *Unit*, and an element function that returns the set *s*. Note that *any* non-empty type can be used as the index type.

![] rewrite unfold\_sing : {s}  $\longleftrightarrow$  (collect x : Unit. s)

### 4.13.4 Rules

#### Well-formedness

The singleton {s} is well-formed if *s* is a set.

\*[1, 34] rule sing\_isset { | intro [] | } :  
[·]  $\langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$   
[ext]  $\langle \Gamma \rangle \vdash \{s\} \text{ set}$

#### Introduction and elimination

The *only* element of the singleton set {s} is the set *s*.

\*[4, 37] rule sing\_member\_elim { | elim [] | }  $\Gamma$  :  
[ext]  $\langle \Gamma \rangle; x : y \in s \{s\}; \langle \Delta[x] \rangle; w : eq(y; s) \vdash T[x] \longrightarrow$   
[ext]  $\langle \Gamma \rangle; x : y \in s \{s\}; \langle \Delta[x] \rangle \vdash T[x]$   
\*[2, 74] rule sing\_member\_intro { | intro [] | } :  
[wf] [·]  $\langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
[wf] [·]  $\langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
[ext]  $\langle \Gamma \rangle \vdash eq(s_1; s_2) \longrightarrow$   
[ext]  $\langle \Gamma \rangle \vdash s_1 \in s \{s_2\}$

#### Functionality

The singleton is functional in its set argument.

\*[13, 523] **rule** `sing_fun`  $\{|$  `intro [] |}  $\}$  :  

$$\frac{[ext] \langle \Gamma \rangle \vdash \forall z. s[z] \text{ fun\_set} \longrightarrow}{[ext] \langle \Gamma \rangle \vdash \forall z. \{s[z]\} \text{ fun\_set}}$$`

## 4.14 Czf\_itt\_union module

The `Czf_itt_union` module gives two definitions of set unions. The  $s_1 \cup s_2$  is the binary union of two sets  $s_1$  and  $s_2$ . The elements of the binary union include the elements in  $s_1$  as well as the elements of  $s_2$ . The general union  $\bigcup s$  represents the union of all the elements in  $s$ . A set  $x$  is a member of the union  $\bigcup s$  if it is a member of any element of  $s$ .

The binary union is defined for sets  $s_1 = \text{collect}(x_1 \in T_1. f_1[x_1])$  and  $s_2 = \text{collect}(x_2 \in T_2. f_2[x_2])$  as a set with the disjoint union index type  $T_1 + T_2$  defined in the `Itt_union` module.

$$\begin{aligned} & \text{collect}(x_1 \in T_1. f_1[x_1]) \cup \text{collect}(x_2 \in T_2. f_2[x_2]) \equiv \\ & \text{collect}(x \in T_1 + T_2. \mathbf{match} \ x \ \mathbf{with} \ \mathit{inl}(u) \rightarrow f_1[u] \mid \mathit{inr}(v) \rightarrow f_2[v]) \end{aligned}$$

The definition of the general union  $\bigcup s$  is more difficult. First, suppose the set  $s$  has the form  $\text{collect}(x \in T. f[x])$ , and that the elements of  $s$  have the form  $\text{collect}(y \in S_x. g_x[y])$  for each term  $x$  in the index type  $T$ . The index type of the union is formed as the dependent product space  $x: T \times S_x$ . Given a pair  $(x, y)$  in the index type, the elements are  $g_x[y]$ . Effectively, the union has the following definition.

$$\begin{aligned} & \bigcup \text{collect}(x \in T. \lambda x. \text{collect}(y \in S[x]. g[x; y])) \equiv \\ & \text{collect}((x, y) \in z: T \times S[z]. g[x; y]) \end{aligned}$$

### 4.14.1 Parents

**extends** `Czf_itt_dexists`  
**extends** `Czf_itt_subset`

### 4.14.2 Terms

**declare** `Czf_itt_union!union` $\{ 's1; 's2 \}$  (**displayed as**  $s_1 \cup s_2$ )  
**declare** `Czf_itt_union!union` $\{ 's1 \}$  (**displayed as**  $\bigcup s_1$ )

### 4.14.3 Rewrites

The binary union  $s_1 \cup s_2$  is defined by simultaneous induction on the set arguments. The index type of the result is the disjoint union of their index types.

```

![] rewrite unfold_bunion :
  ( $s_1 \cup s_2$ )  $\longleftrightarrow$ 
    (let set( $a_1, f_1$ ). $g_1 = s_1$  in
     let set( $a_2, f_2$ ). $g_2 = s_2$  in
     collect  $x : (a_1 + a_2)$ .
     match  $x$  with inl  $z - > (f_1 z) \mid$  inr  $z - > (f_2 z)$ )
*[1, 137] rewrite reduce_bunion {[] reduce []} :
  (collect  $x_1 : t_1. f_1[x_1] \cup s$  collect  $x_2 : t_2. f_2[x_2]$ )  $\longleftrightarrow$ 
    (collect  $x : (t_1 + t_2)$ .
     match  $x$  with inl  $z - > f_1[z] \mid$  inr  $z - > f_2[z]$ )

```

The general union is formed by a nested induction on the set argument. This term formalizes the informal discussion above.

```

![] rewrite unfold_union :
  ( $\cup s$ )  $\longleftrightarrow$ 
    (let set( $a_1, f_1$ ). $g_1 = s$  in
     collect  $y : (x : a_1 \times (let set(a_2, f_2).g_2 = f_1 x$  in  $a_2))$ .
     (match  $y$  with
      ( $u, v$ )  $\rightarrow$ 
      (let set( $a_3, f_3$ ). $g_3 = f_1 u$  in  $f_3 v$ )))

```

### 4.14.4 Rules

#### Well-formedness

Both forms of union are well-formed if their arguments are sets.

```

*[8, 287] rule bunion_isset {[] intro [] []} :
  [wf] [·]  $\langle \Gamma \rangle \vdash s_1$  set  $\rightarrow$ 
  [wf] [·]  $\langle \Gamma \rangle \vdash s_2$  set  $\rightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (s_1 \cup s_2)$  set
*[15, 387] rule union_isset {[] intro [] []} :
  [wf] [·]  $\langle \Gamma \rangle \vdash s_1$  set  $\rightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (\cup s_1)$  set

```

#### Introduction

The binary union  $s_1 \cup s_2$  has two membership introduction forms for an argument set  $x$ ; the set  $x$  may be a member of  $s_1$  or it may be a member of  $s_2$ .

\*[10, 538] **rule bunion\_member\_intro\_left**  
 $\{ | \text{intro} [(\text{SelectOption } 1)] | \} :$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s_{s_1} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s (s_1 \cup s_2)$

\*[10, 538] **rule bunion\_member\_intro\_right**  
 $\{ | \text{intro} [(\text{SelectOption } 2)] | \} :$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s_{s_2} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s (s_1 \cup s_2)$

A set  $x$  is in the general union  $\bigcup s$  if there is some element  $y \in_s s$  for which  $x \in_s y$ .

\*[42, 1587] **rule union\_member\_intro**  $\{ | \text{intro} [] | \} :$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \exists y \in s s. x \in_s y \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s (\bigcup s)$

## Elimination

The elimination form for membership in the binary union performs a case analysis on membership in the two sets in the binary union.

\*[20, 909] **rule bunion\_member\_elim**  $\{ | \text{elim} [] | \} \Gamma :$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle \vdash y \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle; z : y \in s_{s_1} \vdash T[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle; z : y \in s_{s_2} \vdash T[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1 \cup s_2); \langle \Delta[x] \rangle \vdash T[x]$

The elimination form for the general union  $x \in_s \bigcup s$  produces a witness  $z \in_s y$  for which  $x \in_s z$ .

\*[27, 901] **rule union\_member\_elim**  $\{ | \text{elim} [] | \} \Gamma :$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (\bigcup s); \langle \Delta[x] \rangle \vdash y \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (\bigcup s); \langle \Delta[x] \rangle \vdash s \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (\bigcup s); \langle \Delta[x] \rangle; z : \text{set}; u : z \in s s; v : y \in s z \vdash T[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (\bigcup s); \langle \Delta[x] \rangle \vdash T[x]$

The union types are both functional in their arguments.

```
*[18, 1275] rule bunion_fun {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ ∀z.s1[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.s2[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(s1[z] ∪ s2[z]) fun_set
*[15, 857] rule union_fun {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ ∀z.s[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(∪ s [z]) fun_set
```

## 4.15 Czf\_itt\_pair module

The `Czf_itt_pair` module defines the binary pairing constructor  $(s_1, s_2)$ . The pair is derived from the union and singleton.

$$(s_1, s_2) \equiv \{s_1\} \cup \{s_2\}$$

The pair has two elements: the set  $s_1$  and the set  $s_2$ .

### 4.15.1 Parents

```
extends Czf_itt_union
extends Czf_itt_singleton
```

### 4.15.2 Terms

```
declare Czf_itt_pair!pair{'s1; 's2} (displayed as (s1, s2))
```

### 4.15.3 Rewrites

The pair is a union of two singleton sets.

```
![] rewrite unfold_pair : (s1, s2) ←→ ({s1} ∪ s {s2})
```

### 4.15.4 Rules

#### Well-formedness

The pair is a set if both arguments are sets.

\*[1, 39] **rule pair\_isset** { | intro [] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash (s_1, s_2) \text{ set}$

## Introduction

The elements of the pair  $(s_1, s_2)$  are the sets  $s_1$  and  $s_2$ .

\*[2, 66] **rule pair\_member\_intro\_left**  
{ | intro [(SelectOption 1)] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(x; s_1) \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s (s_1, s_2)$   
\*[2, 66] **rule pair\_member\_intro\_right**  
{ | intro [(SelectOption 2)] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash eq(x; s_2) \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash x \in s (s_1, s_2)$

## Elimination

The *only* elements of the pair  $(s_1, s_2)$  are the sets  $s_1$  and  $s_2$ .

\*[10, 112] **rule pair\_member\_elim** { | elim [] | }  $\Gamma$  :  
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle \vdash y \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle; z : eq(y; s_1) \vdash T[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle; z : eq(y; s_2) \vdash T[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : y \in s (s_1, s_2); \langle \Delta[x] \rangle \vdash T[x]$

## Functionality

The pair is functional in both its arguments.

\*[1, 39] **rule pair\_fun** { | intro [] | } :  
 $[ext] \langle \Gamma \rangle \vdash \forall z. s_1[z] \text{ fun\_set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \forall z. s_2[z] \text{ fun\_set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \forall z. (s_1[z], s_2[z]) \text{ fun\_set}$

## 4.16 Czf\_itt\_nat module

The `Czf_itt_nat` module defines the infinite set  $\omega$ . We use the definition of  $\omega$  as the definition of the natural numbers. The `zero` term represents the zero, and the `succ` term  $s(i)$  represents  $i + 1$ . The construction is the usual ordinal construction.

$$\begin{aligned} 0_s &\equiv \{\} \\ s(0_s) &\equiv 0_s \cup \{0_s\} \\ nil &\vdots nil \\ s(i) &\equiv i \cup \{i\} \end{aligned}$$

The definition of the  $\omega$  set uses the `list` type as an index type, and it codes the elements using the list induction combinator.

$$\omega \equiv \text{collect}(l \in \text{list}(\text{Unit}).\mathbf{match} \ l \ \mathbf{with} \ \text{cons}(h, t).g \rightarrow s(g))$$

We also define the usual ordering  $i <_s j$  on numbers (this is just a membership judgment).

### 4.16.1 Parents

```
extends Czf_itt_member
extends Czf_itt_singleton
extends Czf_itt_union
extends Czf_itt_empty
extends Czf_itt_implies
```

### 4.16.2 Terms

```
declare Czf_itt_nat!inf (displayed as  $\mathbb{N}_s$ )
declare Czf_itt_nat!zero (displayed as  $0_s$ )
declare Czf_itt_nat!succ{'i} (displayed as  $s(i)$ )
declare Czf_itt_nat!lt{'i; 'j} (displayed as  $i < j$ )
```

### 4.16.3 Rewrites

The following four rewrites give the definition of the natural numbers. The  $\omega$  set itself is an ordinal construction.

```
![] rewrite unfold_zero :  $0_s \longleftrightarrow \{\}$ 
![] rewrite unfold_succ :  $s(i) \longleftrightarrow (i \cup s \{i\})$ 
![] rewrite unfold_inf :
   $\mathbb{N}_s \longleftrightarrow$ 
```

(collect l : Unit List.  
 match l with [] -> {} | h :: t.g -> s(g))  
 ![] rewrite unfold\_lt : (i < j) ↔ (i ∈ s j)

#### 4.16.4 Rules

##### Well-formedness

Zero is a set, and the successor of *any* set is a set. Infinity is also a set.

\*[1, 11] **rule zero\_isset** {| intro [] |} :  
 [ext] ⟨Γ⟩ ⊢ 0s set  
 \*[1, 30] **rule succ\_isset** {| intro [] |} :  
 [·] ⟨Γ⟩ ⊢ i set →  
 [ext] ⟨Γ⟩ ⊢ s(i) set  
 \*[4, 86] **rule inf\_isset** {| intro [] |} :  
 [ext] ⟨Γ⟩ ⊢ ℕs set  
 \*[1, 30] **rule succ\_fun** {| intro [] |} :  
 [ext] ⟨Γ⟩ ⊢ ∀z.s[z] fun\_set →  
 [ext] ⟨Γ⟩ ⊢ ∀z.s(s[z]) fun\_set

The zero is also a number, and the successor is a number if it's argument is a number.

\*[3, 199] **rule zero\_isnat** {| intro [] |} :  
 [ext] ⟨Γ⟩ ⊢ 0s ∈ s ℕs  
 \*[16, 537] **rule succ\_isnat** {| intro [] |} :  
 [wf] [ext] ⟨Γ⟩ ⊢ i set →  
 [ext] ⟨Γ⟩ ⊢ i ∈ s ℕs →  
 [ext] ⟨Γ⟩ ⊢ s(i) ∈ s ℕs

##### Induction

The induction rule performs induction on an arbitrary number expression in the goal. The goal must be functional over *all* sets. The induction has the usual cases: to prove  $C[i]$  for a number  $i$ , prove  $C[0_s]$ , and given  $C[i]$ , prove  $C[s(i)]$ . The proof of this rule is rather extensive, but it is derived from usual inductive reasoning on sets.

\*[62, 1740] **rule nat\_elim** bind(z.C[z]) i :  
 [wf] [ext] ⟨Γ⟩ ⊢ i ∈ S ℕs →  
 [wf] [ext] ⟨Γ⟩ ⊢ ∀z.C[z] fun\_prop →  
 [base] [ext] ⟨Γ⟩ ⊢ C[0s] →  
 [step] [ext] ⟨Γ⟩; j : set; u : j ∈ s ℕs; v : C[j] ⊢ C[s(j)] →  
 [ext] ⟨Γ⟩ ⊢ C[i]

## Functionality

The  $i <_s j$  relation is functional on its arguments, and it is also a restricted proposition.

\*[1, 21] **rule lt\_fun** { | intro [] | } :  
 $[ext] \langle \Gamma \rangle \vdash \forall z. s_1[z] \text{ fun\_set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \forall z. s_2[z] \text{ fun\_set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash \forall z. (s_1[z] < s_2[z]) \text{ fun\_prop}$   
 \*[1, 21] **rule lt\_restricted** { | intro [] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash < (s_1 < s_2) >$

## Zero

The following three rules characterize the **zero** set. Zero is a number that is smaller than every successor.

The rules play a crucial role in forcing the  $\omega$  set to be infinite. Induction by itself is not sufficient, because the induction rule is valid if all the numbers are equal. The zero rules state that the zero term, at least, is different from any successor.

\*[6, 239] **rule zero\_member\_intro** { | intro [] | } :  
 $[wf] [ext] \langle \Gamma \rangle \vdash i \text{ set} \longrightarrow$   
 $[wf] [ext] \langle \Gamma \rangle \vdash i \in s \mathbb{N}s \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash 0s < s(i)$   
 \*[1, 7] **rule zero\_member\_elim** { | elim [] | }  $\Gamma$ :  
 $[wf] [\cdot] \langle \Gamma \rangle; x : i < 0s; \langle \Delta[x] \rangle \vdash i \in s \mathbb{N}s \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : i < 0s; \langle \Delta[x] \rangle \vdash T[x]$

## Successor

The following two rules characterize the successor. The relation  $s(i) <_s s(j)$  is true if-and-only-if  $i <_s j$ . With these final rules, we can show that  $i \neq s(i)$ , so the set  $\omega$  is actually infinite.

\*[13, 1079] **rule succ\_member\_intro1** { | intro [] | } :  
 $[wf] [\cdot] \langle \Gamma \rangle \vdash i \text{ set} \longrightarrow$   
 $[wf] [\cdot] \langle \Gamma \rangle \vdash j \text{ set} \longrightarrow$   
 $[wf] [ext] \langle \Gamma \rangle \vdash i \in s \mathbb{N}s \longrightarrow$   
 $[wf] [ext] \langle \Gamma \rangle \vdash j \in s \mathbb{N}s \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash i < j \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash s(i) < s(j)$   
 \*[28, 1873] **rule succ\_member\_elim1** { | elim [] | }  $\Gamma$ :  
 $[wf] [\cdot] \langle \Gamma \rangle; x : s(i) < s(j); \langle \Delta[x] \rangle \vdash i \text{ set} \longrightarrow$

$$\begin{array}{l}
[wf] [\cdot] \langle \Gamma \rangle; x : s(i) < s(j); \langle \Delta[x] \rangle \vdash j \text{ set} \longrightarrow \\
[wf] [ext] \langle \Gamma \rangle; x : s(i) < s(j); \langle \Delta[x] \rangle \vdash j \in s \mathbb{N}s \longrightarrow \\
[ext] \langle \Gamma \rangle; x : s(i) < s(j); \langle \Delta[x] \rangle; w : i < j \vdash T[x] \longrightarrow \\
[ext] \langle \Gamma \rangle; x : s(i) < s(j); \langle \Delta[x] \rangle \vdash T[x]
\end{array}$$

### 4.16.5 Tactics

`natIndT` The (`natIndT t`) tactic performs induction on the expression  $t$ , which must be a well-formed number.

## 4.17 Czf\_itt\_rel module

The `Czf_itt_rel` module defines Aczel's *collection* scheme :

$$\mathbb{B}(x \in a, y \in b.P[x; y]) \equiv (\forall x \in_s a. \exists y \in_s b. \phi \wedge \forall y \in_s b. \exists x \in_s a. \phi).$$

There are no rules in this module, except for well-formedness. The `rel` term is just a definition.

### 4.17.1 Parents

```

extends Czf_itt_dall
extends Czf_itt_dexists

```

### 4.17.2 Terms

```

declare
  Czf_itt_rel!rel{a, b. P['a; 'b]; 's1; 's2}
  (displayed as  $\mathbb{B}a \in_{s_1}, b \in_{s_2}. P[a; b]$ )

```

### 4.17.3 Rewrites

```

![] rewrite unfold_rel :
  ( $\mathbb{B}a \in_{s_1}, b \in_{s_2}. P[a; b]$ )  $\longleftrightarrow$ 
  (( $\forall x \in_s s_1. \exists y \in_s s_2. P[x; y]$ )  $\wedge$  ( $\forall y \in_s s_2. \exists x \in_s s_1. P[x; y]$ ))

```

#### 4.17.4 Rules

The `rel` term is well-formed if the proposition  $P$  is well-formed, and if the arguments  $s_1$  and  $s_2$  are sets.

```
*[1, 213] rule rel.type { | intro [] | } :
  [·] ⟨Γ⟩; u : set; v : set ⊢ P[u; v] Type →
  [·] ⟨Γ⟩ ⊢ s1 set →
  [·] ⟨Γ⟩ ⊢ s2 set →
  [ext] ⟨Γ⟩ ⊢ (ℬx ∈ s1, y ∈ s2. P[x; y]) Type
```

### 4.18 Czf\_itt\_power module

The `Czf_itt_power` module defines *subset* collection. Contrary to the name, the subset collection is *not* a power set, but it has some similarities. The subset collection constructor has the form  $\mathbb{P}_{s_1} \rightarrow s_1$ , where  $s_1$  and  $s_2$  are sets. If the canonical forms are  $s_1 = \text{collect}(x_1 \in T_1.f_1[x_1])$  and  $s_2 = \text{collect}(x_2 \in T_2.f_2[x_2])$ , the elements of the power set are the subsets of  $s_2$  that are defined by the images of the computable functions in the space  $T_1 \rightarrow T_2$ .

$$\mathbb{P}_{s_1} \rightarrow s_1 \equiv \text{collect}(g \in T_1 \rightarrow T_2.\text{collect}(x \in T_1.f_2[g(x)]))$$

There is only one significant rule in this module: the axiom of subset collection.

#### 4.18.1 Parents

```
extends Czf_itt_subset
extends Czf_itt_rel
```

#### 4.18.2 Terms

```
declare Czf_itt_power!power{'s1; 's2} (displayed as  $\mathbb{P}(s_1 \rightarrow s_2)$ )
```

#### 4.18.3 Rewrites

The subset collection is defined by simultaneous induction on the two set arguments.

```
![] rewrite unfold_scoll :
   $\mathbb{P}(s_1 \rightarrow s_2) \longleftrightarrow$ 
    (let set(T1, f1).g1 = s1 in
     let set(T2, f2).g2 = s2 in
```

$$\begin{array}{l}
\text{collect } x : T_1 \rightarrow T_2. \text{ collect } y : T_1. f_2 (x y) \\
*[1, 135] \text{ rewrite reduce\_scoll } \{| \text{ reduce } |\} : \\
\mathbb{P}((\text{collect } x_1 : T_1. f_1[x_1]) \rightarrow (\text{collect } x_2 : T_2. f_2[x_2])) \longleftrightarrow \\
(\text{collect } x : T_1 \rightarrow T_2. \text{ collect } y : T_1. f_2[(x y)])
\end{array}$$

#### 4.18.4 Rules

##### Well-formedness

The subset collection is well-formed if its arguments are sets.

$$\begin{array}{l}
*[7, 248] \text{ rule power\_isset1 } \{| \text{ intro } [] |\} : \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \mathbb{P}(s_1 \rightarrow s_2) \text{ set}
\end{array}$$

##### The subset collection axiom

There is an element of the power set for each computable function  $s_1 \rightarrow s_2$ .

$$\begin{array}{l}
*[56, 5076] \text{ rule power\_thm } \text{bind}(x.\text{bind}(y.P[x; y])) a b : \\
[wf] [\cdot] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle \vdash b \text{ set} \longrightarrow \\
[wf] [\cdot] \langle \Gamma \rangle; x : \text{set}; y : \text{set} \vdash P[x; y] \text{Type} \longrightarrow \\
[wf] [ext] \langle \Gamma \rangle; x : \text{set} \vdash \forall z. P[x; z] \text{ fun\_prop} \longrightarrow \\
[wf] [ext] \langle \Gamma \rangle; x : \text{set} \vdash \forall z. P[z; x] \text{ fun\_prop} \longrightarrow \\
[antecedent] [ext] \langle \Gamma \rangle; x : \text{set}; u : x \in s a \vdash \exists y \in s b. P[x; y] \longrightarrow \\
[ext] \langle \Gamma \rangle; z : \text{set}; u : z \in s \mathbb{P}(a \rightarrow b); v : \mathbb{B}x \in a, y \in z. P[x; y] \vdash C \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash C
\end{array}$$

#### 4.18.5 Tactics

**powerT** The (**powerT**  $t$ ) tactic applies the **power\_thm** rule. The argument  $t$  is a **rel** term  $\mathbb{B}(x \in a, y \in b. P[x, y])$  that specifies the proposition  $P$  and the sets  $a$  and  $b$  that are the arguments to the **power\_thm**.

### 4.19 Czf\_itt\_axioms module

The **Czf\_itt\_axioms** defines the remaining axioms of the set theory as axioms. This includes the set induction scheme, and the strong collection.

### 4.19.1 Parents

The `Czf_itt_axiom` module includes the entire logical part of the theory, as well as the definition of the `rel` (for use in the definition of the strong collection theorem).

```
extends Czf_itt_true
extends Czf_itt_false
extends Czf_itt_and
extends Czf_itt_or
extends Czf_itt_implies
extends Czf_itt_all
extends Czf_itt_exists
extends Czf_itt_sall
extends Czf_itt_sexists
extends Czf_itt_dall
extends Czf_itt_dexists
extends Czf_itt_rel
```

### 4.19.2 Rules

#### Set induction

The set induction rule formalizes the induction scheme. A goal  $P[z]$  can be proven for a set  $z$  if it can be proven for an arbitrary set  $x$ , given that it holds on all the elements.

The proof of induction follows directly from  $W$ -type induction.

```
*[14, 402] rule set_induction :
  [ext]  $\langle \Gamma \rangle ; x : set \vdash P[x] \mathbf{Type} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash \forall z. P[z] \mathit{fun\_prop} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle ; x : set ; w : \forall z \in s. x. P[z] \vdash P[x] \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash \forall_s z. P[z]$ 
```

#### Strong Collection

The strong collection axiom states that for every proof of a forall/exists formula  $\forall x \in_s s_1. \exists_s s_2. P[x.y]$ , there is a set  $s_3$  that contains the collection of sets that were chosen by the existential. The reason that the collection is not defined as a set constructor is that the proof of the forall/exists formula is part of the construction. If the set  $s_1$  has canonical for  $s_1 = \mathit{collect}(x \in T. f[x])$ , the proof provides a witness that inhabits the function space  $T \rightarrow set$ . The canonical form of the proof is a lambda-function  $\lambda x. s_x$ , which can be used to form the set collection  $\mathit{collect}(x \in T. s_x)$ .

```
*[37, 2268] rule collection  $s_1 \mathit{bind}(x. \mathit{bind}(y. P[x; y]))$ :
```

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash s_1 \text{ set} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; x : \text{set}; y : \text{set} \vdash P[x; y] \text{Type} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \forall x \in s \ s_1. \exists y. P[x; y] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; s_2 : \text{set}; w : \mathbb{B}x \in s_1, y \in s_2. P[x; y] \vdash C \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash C
\end{array}$$

### Subset collection

The `Czf_itt_power` module defines the subset collection set constructor  $\mathbb{P}s_1 \rightarrow s_1$ . For completeness, we reprove the axiom form of the subset collection.

```

* $[43, 2580]$  rule subset_collection a b bind(u.bind(x.bind(y.P[u;
      x;
      y]])):
  [ext]  $\langle \Gamma \rangle \vdash a \text{ set} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash b \text{ set} \longrightarrow$ 
  [ $\cdot$ ]  $\langle \Gamma \rangle; u : \text{set}; x : \text{set}; y : \text{set} \vdash P[u; x; y] \text{Type} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle; u : \text{set}; x : \text{set} \vdash \forall y. P[u; x; y] \text{fun\_prop} \longrightarrow$ 
  [ext]
  1.  $\langle \Gamma \rangle$ 
  2.  $w :$ 
   $\exists_s c. (\forall_s u. ((\forall x \in s \ a. \exists y \in s \ b. P[u; x; y])$ 
     $\rightarrow (\exists z \in s \ c. (\mathbb{B}x \in a, y \in z. P[u; x; y])))$ 
   $\vdash C \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash C$ 

```

## 4.20 Czf\_itt\_group module

The `Czf_itt_group` module defines groups. Each group is assigned a label, such as  $g$ . The predicate  $group(g)$  is used to represent “ $g$  is a group”. The carrier set, operation, identity, and inverse of the group are defined as  $car(g)$ ,  $op(g, a, b)$ ,  $id(g)$ , and  $inv(g, a)$  respectively. *Axioms* are used to describe a group, such as the axioms about the closure property, the axiom about associativity, the axioms about the identity and inverse. Any group should satisfy these axioms; all properties of groups are derived constructively from the axioms of groups and the axioms of set theory.

### 4.20.1 Parents

```

extends Itt_record_label0
extends Czf_itt_dall

```

## 4.20.2 Terms

```
declare Czf_itt_group!group{'g'} (displayed as  $g$  group)
declare Czf_itt_group!car{'g'} (displayed as  $car(g)$ )
declare Czf_itt_group!op{'g; 'a; 'b'} (displayed as  $op(g; a; b)$ )
declare Czf_itt_group!id{'g'} (displayed as  $id(g)$ )
declare Czf_itt_group!inv{'g; 'a'} (displayed as  $inv(g; a)$ )
```

## 4.20.3 Axioms

The group is defined by a set of axioms.

### Well-formedness

The `group`, `car`, `op`, `inv`, and `id` are well-formed if the group argument is a label and the set argument is a set (if there is any).

```
# rule group_type { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
  [ext] ⟨Γ⟩ ⊢  $g$  groupType
# rule car_isset { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
  [ext] ⟨Γ⟩ ⊢  $car(g)$  set
# rule op_isset { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
  [·] ⟨Γ⟩ ⊢  $s_1$  set  $\longrightarrow$ 
  [·] ⟨Γ⟩ ⊢  $s_2$  set  $\longrightarrow$ 
  [ext] ⟨Γ⟩ ⊢  $(op(g; s_1; s_2))$  set
# rule id_isset { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
  [ext] ⟨Γ⟩ ⊢  $id(g)$  set
# rule inv_isset { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $s_1$  set  $\longrightarrow$ 
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
  [ext] ⟨Γ⟩ ⊢  $(inv(g; s_1))$  set
```

### Binary operation

Every `op` is a *binary operation* on `car`, which means: first, if  $a$  and  $b$  are in  $car(g)$ , then  $op(g, a, b)$  is *again* in  $car(g)$ ; second, it assigns each ordered pair exactly one element, i.e., `op` is functional in its set arguments.

```
# rule op_closure { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $s_1$  set  $\longrightarrow$ 
  [·] ⟨Γ⟩ ⊢  $s_2$  set  $\longrightarrow$ 
  [·] ⟨Γ⟩ ⊢  $g \in Label \longrightarrow$ 
```

```

[ext] ⟨Γ⟩ ⊢ g group →
[ext] ⟨Γ⟩ ⊢ s1 ∈ s car(g) →
[ext] ⟨Γ⟩ ⊢ s2 ∈ s car(g) →
[ext] ⟨Γ⟩ ⊢ (op(g; s1; s2)) ∈ s car(g)
# rule op_fun {| intro [] |} :
[.] ⟨Γ⟩ ⊢ g ∈ Label →
[ext] ⟨Γ⟩ ⊢ ∀z.s1[z] fun_set →
[ext] ⟨Γ⟩ ⊢ ∀z.s2[z] fun_set →
[ext] ⟨Γ⟩ ⊢ ∀z.(op(g; s1[z]; s2[z])) fun_set

```

### Associativity

Every op is associative.

```

# rule op_assoc1 {| intro [] |} :
[.] ⟨Γ⟩ ⊢ s1 set →
[.] ⟨Γ⟩ ⊢ s2 set →
[.] ⟨Γ⟩ ⊢ s3 set →
[.] ⟨Γ⟩ ⊢ g ∈ Label →
[ext] ⟨Γ⟩ ⊢ g group →
[ext] ⟨Γ⟩ ⊢ s1 ∈ s car(g) →
[ext] ⟨Γ⟩ ⊢ s2 ∈ s car(g) →
[ext] ⟨Γ⟩ ⊢ s3 ∈ s car(g) →
[ext]
1. ⟨Γ⟩
⊢ eq((op(g; op(g; s1; s2); s3)); (op(g; s1; op(g; s2; s3))))

```

### Identity

Each group  $g$  has an identity  $id(g)$  such that for any  $s \in car(g)$ ,  $eq(op(g, id(g), s), s)$ .

```

# rule id_mem {| intro [] |} :
[.] ⟨Γ⟩ ⊢ g ∈ Label →
[ext] ⟨Γ⟩ ⊢ g group →
[ext] ⟨Γ⟩ ⊢ id(g) ∈ s car(g)
# rule id_eq1 {| intro [] |} :
[.] ⟨Γ⟩ ⊢ s set →
[.] ⟨Γ⟩ ⊢ g ∈ Label →
[ext] ⟨Γ⟩ ⊢ g group →
[ext] ⟨Γ⟩ ⊢ s ∈ s car(g) →
[ext] ⟨Γ⟩ ⊢ eq((op(g; id(g); s)); s)

```

## Inverse

Every  $\text{inv}$  is a *unary operation* on  $\text{car}$  such that  $\text{eq}(\text{op}(g, \text{inv}(g, s), s), \text{id}(g))$  for any  $a \in \text{car}(g)$ .

```
# rule inv_fun {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ ∀z.s[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(inv(g; s[z])) fun_set
# rule inv_mem {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s set →
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ g group →
  [ext] ⟨Γ⟩ ⊢ s ∈ s car(g) →
  [ext] ⟨Γ⟩ ⊢ (inv(g; s)) ∈ s car(g)
# rule inv_id1 {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s set →
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ g group →
  [ext] ⟨Γ⟩ ⊢ s ∈ s car(g) →
  [ext] ⟨Γ⟩ ⊢ eq((op(g; inv(g; s); s)); id(g))
```

## 4.20.4 Rules

### Lemmas

If  $\text{group}(g)$ , then

1. if  $s$  is a member of  $\text{car}(g)$  and  $\text{eq}(\text{op}(g, s, s), s)$ , then  $s$  is the identity.
2. the left inverse is also the right inverse.
3. the left identity is also the right identity.

```
*[7, 1113] rule id_judge_elim {| elim [] |} Γ:
  [·] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩ ⊢ s set →
  [·] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩ ⊢ g group →
  [ext] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩ ⊢ s ∈ s car(g) →
  [ext] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩; y : eq(s; id(g)) ⊢ C[x] →
  [ext] ⟨Γ⟩; x : eq((op(g; s; s)); s); ⟨Δ[x]⟩ ⊢ C[x]
*[5, 2451] rule inv_id2 {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s set →
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ g group →
  [ext] ⟨Γ⟩ ⊢ s ∈ s car(g) →
  [ext] ⟨Γ⟩ ⊢ eq((op(g; s; inv(g; s))); id(g))
```

\*[4, 709] **rule id\_eq2**  $\{ | \text{intro } [] \} :$   
 $[\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 $[\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$   
 $[\text{ext}] \langle \Gamma \rangle \vdash s \in s \text{ car}(g) \longrightarrow$   
 $[\text{ext}] \langle \Gamma \rangle \vdash \text{eq}((\text{op}(g; s; \text{id}(g))); s)$

## Theorems

The left and right cancellation laws.

\*[6, 2507] **rule cancel1**  $\Gamma :$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3))); \langle \Delta[x] \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3))); \langle \Delta[x] \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3))); \langle \Delta[x] \rangle \vdash s_3 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3))); \langle \Delta[x] \rangle \vdash g \in \text{Label} \longrightarrow$   
 $[\text{ext}] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3))); \langle \Delta[x] \rangle \vdash g \text{ group} \longrightarrow$   
 $[\text{ext}]$   
 1.  $\langle \Gamma \rangle$   
 2.  $x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3)))$   
 3.  $\langle \Delta[x] \rangle$   
 $\vdash s_1 \in s \text{ car}(g) \longrightarrow$   
 $[\text{ext}]$   
 1.  $\langle \Gamma \rangle$   
 2.  $x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3)))$   
 3.  $\langle \Delta[x] \rangle$   
 $\vdash s_2 \in s \text{ car}(g) \longrightarrow$   
 $[\text{ext}]$   
 1.  $\langle \Gamma \rangle$   
 2.  $x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3)))$   
 3.  $\langle \Delta[x] \rangle$   
 $\vdash s_3 \in s \text{ car}(g) \longrightarrow$   
 $[\text{ext}]$   
 1.  $\langle \Gamma \rangle$   
 2.  $x : \text{eq}((\text{op}(g; s_1; s_2)); (\text{op}(g; s_1; s_3)))$   
 3.  $\langle \Delta[x] \rangle$   
 $\vdash \text{eq}(s_2; s_3)$   
 \*[6, 2507] **rule cancel2**  $\Gamma :$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_3)); (\text{op}(g; s_2; s_3))); \langle \Delta[x] \rangle \vdash s_1 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_3)); (\text{op}(g; s_2; s_3))); \langle \Delta[x] \rangle \vdash s_2 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_3)); (\text{op}(g; s_2; s_3))); \langle \Delta[x] \rangle \vdash s_3 \text{ set} \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_3)); (\text{op}(g; s_2; s_3))); \langle \Delta[x] \rangle \vdash g \in \text{Label} \longrightarrow$   
 $[\text{ext}] \langle \Gamma \rangle; x : \text{eq}((\text{op}(g; s_1; s_3)); (\text{op}(g; s_2; s_3))); \langle \Delta[x] \rangle \vdash g \text{ group} \longrightarrow$   
 $[\text{ext}]$

1.  $\langle \Gamma \rangle$
2.  $x : eq((op(g; s_1; s_3)); (op(g; s_2; s_3)))$
3.  $\langle \Delta[x] \rangle$

 $\vdash s_1 \in s \text{ car}(g) \longrightarrow$ 

[*ext*]

1.  $\langle \Gamma \rangle$
2.  $x : eq((op(g; s_1; s_3)); (op(g; s_2; s_3)))$
3.  $\langle \Delta[x] \rangle$

 $\vdash s_2 \in s \text{ car}(g) \longrightarrow$ 

[*ext*]

1.  $\langle \Gamma \rangle$
2.  $x : eq((op(g; s_1; s_3)); (op(g; s_2; s_3)))$
3.  $\langle \Delta[x] \rangle$

 $\vdash s_3 \in s \text{ car}(g) \longrightarrow$ 

[*ext*]

1.  $\langle \Gamma \rangle$
2.  $x : eq((op(g; s_1; s_3)); (op(g; s_2; s_3)))$
3.  $\langle \Delta[x] \rangle$

 $\vdash eq(s_1; s_2)$ 

Unique identity (left and right).

\*[2, 485] **rule unique\_id1** :

$$[\cdot] \langle \Gamma \rangle \vdash e_2 \text{ set} \longrightarrow$$

$$[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash e_2 \in s \text{ car}(g) \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash \forall s \in s \text{ car}(g).eq((op(g; e_2; s)); s) \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash eq(e_2; id(g))$$

\*[2, 491] **rule unique\_id2** :

$$[\cdot] \langle \Gamma \rangle \vdash e_2 \text{ set} \longrightarrow$$

$$[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash e_2 \in s \text{ car}(g) \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash \forall s \in s \text{ car}(g).eq((op(g; s; e_2)); s) \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash eq(e_2; id(g))$$

Unique inverse (left and right).

\*[3, 706] **rule unique\_inv1** { | intro [] | } :

$$[\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow$$

$$[\cdot] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow$$

$$[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$$

$$[ext] \langle \Gamma \rangle \vdash s \in s \text{ car}(g) \longrightarrow$$

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash s_2 \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq((op(g; s_2; s)); id(g)) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_2; (inv(g; s))) \\
*[3, 657] \text{ rule unique\_inv2 } \{ | \text{ intro } [] \} : \\
[.] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash s_2 \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash s_2 \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq((op(g; s; s_2)); id(g)) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(s_2; (inv(g; s)))
\end{array}$$

Unique solution.

$$\begin{array}{l}
*[4, 1772] \text{ rule unique\_sol1 } \{ | \text{ intro } [] \} : \\
[.] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash b \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash x \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash b \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash x \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq((op(g; a; x)); b) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(x; (op(g; inv(g; a); b))) \\
*[4, 1772] \text{ rule unique\_sol2 } \{ | \text{ intro } [] \} : \\
[.] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash b \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash y \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash b \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash y \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq((op(g; y; a)); b) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(y; (op(g; b; inv(g; a))))
\end{array}$$

Inverse simplification.

$$\begin{array}{l}
*[4, 2127] \text{ rule inv\_simplify } \{ | \text{ intro } [] \} : \\
[.] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash b \text{ set} \longrightarrow \\
[.] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow
\end{array}$$

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash b \in s \text{ car}(g) \longrightarrow \\
[ext] \\
1. \langle \Gamma \rangle \\
\vdash eq((inv(g; op(g; a; b))); (op(g; inv(g; b); inv(g; a))))
\end{array}$$

### 4.20.5 Tactics

`groupCancelLeftT`, `groupCancelRightT`, `uniqueInvLeftT`, `uniqueInvRightT`

The `groupCancelLeftT` tactic applies the `cancel1` rule, which infers that  $a$  and  $b$  are equal from the fact that  $c * a$  is equal to  $c * b$ . The `groupCancelRightT` tactic applies the `cancel2` rule, which infers that  $a$  and  $b$  are equal from the fact that  $a * c$  equals  $b * c$ . The `uniqueInvLeftT` and `uniqueInvRightT` tactics apply the `unique_inv_elim1` and `unique_inv_elim2` rules and prove  $x$  is the inverse of  $y$  from the fact that  $y * x$  or  $x * y$  is the identity respectively.

## 4.21 Czf\_itt\_kleingroup module

The `Czf_itt_kleingroup` module defines the klein 4-group, which contains 4 elements  $e, a, b, c$ , and whose group table is:

	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

The klein 4-group is assigned a label `klein4`. Its carrier set, operation, identity, and inverse are all rewritten in terms of its elements `k0`, `k1`, `k2`, and `k3`. By showing all the axioms for groups are satisfied, we prove that `klein4` is a group.

### 4.21.1 Parents

```

extends Czf_itt_group
extends Czf_itt_singleton
extends Czf_itt_union

```

## 4.21.2 Terms

```
declare Czf_itt_kleingroup!klein4 (displayed as klein4)
declare Czf_itt_kleingroup!k0 (displayed as k0)
declare Czf_itt_kleingroup!k1 (displayed as k1)
declare Czf_itt_kleingroup!k2 (displayed as k2)
declare Czf_itt_kleingroup!k3 (displayed as k3)
```

## 4.21.3 Rewrites

The definition of the klein 4-group.

```
![] rewrite unfold_klein4_car :
  car(klein4)  $\longleftrightarrow$  ( $\{k0\} \cup s \{k1\} \cup s \{k2\} \cup s \{k3\}$ )
![] rewrite unfold_klein4_op00 : (op(klein4; k0; k0))  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_op01 : (op(klein4; k0; k1))  $\longleftrightarrow$  k1
![] rewrite unfold_klein4_op02 : (op(klein4; k0; k2))  $\longleftrightarrow$  k2
![] rewrite unfold_klein4_op03 : (op(klein4; k0; k3))  $\longleftrightarrow$  k3
![] rewrite unfold_klein4_op10 : (op(klein4; k1; k0))  $\longleftrightarrow$  k1
![] rewrite unfold_klein4_op11 : (op(klein4; k1; k1))  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_op12 : (op(klein4; k1; k2))  $\longleftrightarrow$  k3
![] rewrite unfold_klein4_op13 : (op(klein4; k1; k3))  $\longleftrightarrow$  k2
![] rewrite unfold_klein4_op20 : (op(klein4; k2; k0))  $\longleftrightarrow$  k2
![] rewrite unfold_klein4_op21 : (op(klein4; k2; k1))  $\longleftrightarrow$  k3
![] rewrite unfold_klein4_op22 : (op(klein4; k2; k2))  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_op23 : (op(klein4; k2; k3))  $\longleftrightarrow$  k1
![] rewrite unfold_klein4_op30 : (op(klein4; k3; k0))  $\longleftrightarrow$  k3
![] rewrite unfold_klein4_op31 : (op(klein4; k3; k1))  $\longleftrightarrow$  k2
![] rewrite unfold_klein4_op32 : (op(klein4; k3; k2))  $\longleftrightarrow$  k1
![] rewrite unfold_klein4_op33 : (op(klein4; k3; k3))  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_id : id(klein4)  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_inv0 : (inv(klein4; k0))  $\longleftrightarrow$  k0
![] rewrite unfold_klein4_inv1 : (inv(klein4; k1))  $\longleftrightarrow$  k1
![] rewrite unfold_klein4_inv2 : (inv(klein4; k2))  $\longleftrightarrow$  k2
![] rewrite unfold_klein4_inv3 : (inv(klein4; k3))  $\longleftrightarrow$  k3
```

## 4.21.4 Rules

### Well-formedness

The `klein4` is a label and `k0`, `k1`, `k2`, `k3` are all sets. These are axioms.

```
# rule klein4_label { | intro [] | } :
  [ext]  $\langle \Gamma \rangle \vdash$  klein4  $\in$  Label
# rule k0_isset { | intro [] | } :
  [ext]  $\langle \Gamma \rangle \vdash$  k0 set
```

```

# rule k1_isset {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k1 set
# rule k2_isset {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k2 set
# rule k3_isset {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k3 set

```

The  $car(klein4)$ ,  $id(klein4)$  are well-formed; the  $op(klein4, s_1, s_2)$  is well-formed if its set arguments are both sets; the  $inv(klein4, s)$  is well-formed if its set argument is a set.

```

*[1, 18] rule klein4_car_isset {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ car(klein4) set
*[1, 28] rule klein4_op_isset {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [·] ⟨Γ⟩ ⊢ s2 set →
  [ext] ⟨Γ⟩ ⊢ (op(klein4; s1; s2)) set
*[1, 18] rule klein4_id_isset {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ id(klein4) set
*[1, 23] rule klein4_inv_isset {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [ext] ⟨Γ⟩ ⊢ (inv(klein4; s1)) set

```

### Introduction and elimination for the carrier set

The  $car(klein4)$  contains  $k0$ ,  $k1$ ,  $k2$ ,  $k3$  only.

```

*[2, 160] rule car_klein0 {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k0 ∈ s car(klein4)
*[2, 236] rule car_klein1 {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k1 ∈ s car(klein4)
*[2, 276] rule car_klein2 {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k2 ∈ s car(klein4)
*[2, 276] rule car_klein3 {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ k3 ∈ s car(klein4)
*[1, 380] rule car_klein0_elim {| elim [] |} Γ:
  [·] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩ ⊢ y set →
  [ext] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩; z : eq(y; k0) ⊢ T[x] →
  [ext] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩; z : eq(y; k1) ⊢ T[x] →
  [ext] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩; z : eq(y; k2) ⊢ T[x] →
  [ext] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩; z : eq(y; k3) ⊢ T[x] →
  [ext] ⟨Γ⟩; x : y ∈ s car(klein4); ⟨Δ[x]⟩ ⊢ T[x]

```

## Verification of the group axioms

The `op` for `klein4` is functional and is a mapping.

```
*[1, 28] rule klein4_op_fun {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ ∀z.s1[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.s2[z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(op(klein4; s1[z]; s2[z])) fun_set
*[21, 2797] rule klein4_op_closure {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [·] ⟨Γ⟩ ⊢ s2 set →
  [ext] ⟨Γ⟩ ⊢ s1 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ s2 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ (op(klein4; s1; s2)) ∈ s car(klein4)
```

The `op` for `klein4` is associative.

```
*[85, 47203] rule klein4_op_assoc1 {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [·] ⟨Γ⟩ ⊢ s2 set →
  [·] ⟨Γ⟩ ⊢ s3 set →
  [ext] ⟨Γ⟩ ⊢ s1 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ s2 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ s3 ∈ s car(klein4) →
  [ext]
  1. ⟨Γ⟩
  ⊢
  eq((op(klein4; op(klein4; s1; s2); s3)); (op(klein4; s1; op(klein4; s2; s3))))
```

The axioms for the identity are satisfied.

```
*[1, 13] rule klein4_id_mem {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ id(klein4) ∈ s car(klein4)
*[5, 1241] rule klein4_id_eq1 {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s set →
  [ext] ⟨Γ⟩ ⊢ s ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ eq((op(klein4; id(klein4); s)); s)
```

The axioms for the inverse are satisfied.

```
*[1, 23] rule klein4_inv_fun {| intro [] |} :
  [ext] ⟨Γ⟩ ⊢ ∀z.(inv(klein4; z)) fun_set
*[5, 447] rule klein4_inv_mem {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [ext] ⟨Γ⟩ ⊢ s1 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ (inv(klein4; s1)) ∈ s car(klein4)
```

```

*[5, 1653] rule klein4_inv_id1 {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ s1 set →
  [ext] ⟨Γ⟩ ⊢ s1 ∈ s car(klein4) →
  [ext] ⟨Γ⟩ ⊢ eq((op(klein4; inv(klein4; s1); s1)); id(klein4))

```

## 4.22 Czf\_itt\_abel\_group module

The `Czf_itt_abel_group` module defines abelian groups. A group is *abelian* if its binary operation is commutative.

### 4.22.1 Parents

```
extends Czf_itt_group
```

### 4.22.2 Terms

```
declare Czf_itt_abel_group!abel{'g} (displayed as abel(g))
```

### 4.22.3 Rewrites

A group  $g$  is abelian if its operation is commutative.

```

![] rewrite unfold_abel :
  abel(g) ↔
    (g group
     ∧ (∀ a : set
        ∀ b : set
          (a ∈ s car(g)
           → (b ∈ s car(g))
              → eq((op(g; a; b)); (op(g; b; a))))))

```

### 4.22.4 Rules

#### Typehood

The `abel` judgment is well-formed if its argument is a label.

```

*[1, 269] rule abel_type {| intro [] |} :
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ abel(g) Type

```

## Introduction

The proposition  $abel(g)$  is true if it is well-formed,  $g$  is a group, and  $op$  is commutative.

```
*[1, 159] rule abel_intro { | intro [] | } :
  [·] ⟨Γ⟩ ⊢  $g \in Label$  →
  [ext] ⟨Γ⟩ ⊢  $g\ group$  →
  [ext]
  1. ⟨Γ⟩
  2.  $a : set$ 
  3.  $b : set$ 
  4.  $x : a \in s\ car(g)$ 
  5.  $y : b \in s\ car(g)$ 
  ⊢  $eq((op(g; a; b)); (op(g; b; a)))$  →
  [ext] ⟨Γ⟩ ⊢  $abel(g)$ 
```

## 4.23 Czf\_itt\_subgroup module

The `Czf_itt_subgroup` module defines subgroups. A subgroup of a group  $g$  is a group whose carrier is a subset of  $g$  and who shares the same binary operation as  $g$ .

### 4.23.1 Parents

```
extends Czf_itt_group
extends Czf_itt_subset
extends Czf_itt_isect
```

### 4.23.2 Terms

```
declare
  Czf_itt_subgroup!subgroup{'s; 'g}
  (displayed as  $subgroup(s; g)$ )
```

### 4.23.3 Rewrites

A group  $s$  is a subgroup of group  $g$  if the carrier of  $s$  is a subset of that of  $g$  and the operation of  $s$  is the same as that of  $g$ .

```
![] rewrite unfold_subgroup :
   $subgroup(s; g) \longleftrightarrow$ 
```

$$\begin{aligned}
& (s \text{ group} \\
& \wedge g \text{ group} \\
& \wedge (car(s) \subseteq s \text{ car}(g)) \\
& \wedge (\forall a : set \\
& \quad \forall b : set \\
& \quad (a \in s \text{ car}(s) \\
& \quad \rightarrow (b \in s \text{ car}(s) \\
& \quad \rightarrow eq((op(s; a; b)); (op(g; a; b))))))
\end{aligned}$$

#### 4.23.4 Rules

##### Typehood

The  $subgroup(s, g)$  is well-formed if its arguments are labels.

$$\begin{aligned}
& * [1, 336] \text{ rule subgroup\_wf } \{ | \text{ intro } [] \} : \\
& [\cdot] \langle \Gamma \rangle \vdash s \in Label \longrightarrow \\
& [\cdot] \langle \Gamma \rangle \vdash g \in Label \longrightarrow \\
& [ext] \langle \Gamma \rangle \vdash subgroup(s; g) \text{ Type}
\end{aligned}$$

##### Introduction

The proposition  $subgroup(s, g)$  is true if it is well-formed,  $s$  and  $g$  are groups,  $car(s)$  is a subset of  $car(g)$ , and  $op(s, a, b)$  is defined as  $op(g, a, b)$  for  $a, b \in car(s)$ .

$$\begin{aligned}
& * [1, 185] \text{ rule subgroup\_intro } \{ | \text{ intro } [] \} : \\
& [\cdot] \langle \Gamma \rangle \vdash s \in Label \longrightarrow \\
& [\cdot] \langle \Gamma \rangle \vdash g \in Label \longrightarrow \\
& [ext] \langle \Gamma \rangle \vdash s \text{ group} \longrightarrow \\
& [ext] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\
& [ext] \langle \Gamma \rangle \vdash car(s) \subseteq s \text{ car}(g) \longrightarrow \\
& [ext] \\
& 1. \langle \Gamma \rangle \\
& 2. a : set \\
& 3. b : set \\
& 4. x : a \in s \text{ car}(s) \\
& 5. y : b \in s \text{ car}(s) \\
& \vdash eq((op(s; a; b)); (op(g; a; b))) \longrightarrow \\
& [ext] \langle \Gamma \rangle \vdash subgroup(s; g)
\end{aligned}$$

##### Properties

If  $s$  is a subgroup of  $g$ , then

1.  $s$  is closed under the binary operation of  $g$ .
2. the identity of  $s$  is the identity of  $g$ .
3. the inverse of  $a \in \text{car}(s)$  is also the inverse of  $a$  in  $g$ .

\*[3, 804] **rule subgroup\_op** { | intro [] | } :

[.]  $\langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash a \text{ set} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash b \text{ set} \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash a \in s \text{ car}(s) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash b \in s \text{ car}(s) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{subgroup}(s; g) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash (\text{op}(g; a; b)) \in s \text{ car}(s)$

\*[7, 1220] **rule subgroup\_id1** { | intro [] | } :

[.]  $\langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{subgroup}(s; g) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{eq}(\text{id}(s); \text{id}(g))$

\*[2, 444] **rule subgroup\_id2** { | intro [] | } :

[.]  $\langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{subgroup}(s; g) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{id}(g) \in s \text{ car}(s)$

\*[6, 2057] **rule subgroup\_inv1** { | intro [] | } :

[.]  $\langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash a \text{ set} \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash a \in s \text{ car}(s) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{subgroup}(s; g) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{eq}((\text{inv}(s; a)); (\text{inv}(g; a)))$

\*[3, 556] **rule subgroup\_inv2** { | intro [] | } :

[.]  $\langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$   
 [.]  $\langle \Gamma \rangle \vdash a \text{ set} \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash a \in s \text{ car}(s) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash \text{subgroup}(s; g) \longrightarrow$   
 [ext]  $\langle \Gamma \rangle \vdash (\text{inv}(g; a)) \in s \text{ car}(s)$

## Theorems

The intersection group of subgroups  $h_1$  and  $h_2$  of a group  $g$  is again a subgroup of  $g$ .

\*[10, 1723] **rule subgroup\_isect**  $h_1 h_2$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_1; g) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_2; g) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash h \text{ group} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{car}(h) = ( \text{car}(h_1) \cap \text{car}(h_2) ) \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. a : \text{set} \\
3. b : \text{set} \\
4. x : a \in s \text{ car}(h) \\
5. y : b \in s \text{ car}(h) \\
\vdash \text{eq}((\text{op}(h; a; b)); (\text{op}(h_1; a; b))) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h; g)
\end{array}$$

### 4.23.5 Tactics

`subgroupIsectT` The tactic applies the `subgroup_isect` rule and proves group  $h$  is a subgroup of  $g$  if it is the intersection of two subgroups of  $g$ .

## 4.24 Czf\_itt\_group\_power module

The `Czf_itt_group_power` module defines the power operation in a group, i.e., it describes  $x^n = x * x * \dots * x$ .

$x^n$  is defined by induction on  $n$  as

$$\begin{array}{l}
\text{power}(g, z, n) \equiv \\
\text{ind}(n; i, j. \text{op}(g, \text{inv}(g, z), \text{power}(g, z, (n + 1))); \text{id}(g); k, k. \text{op}(g, z, \text{power}(g, z, (n - 1))))
\end{array}$$

### 4.24.1 Parents

`extends` `Czf_itt_group`  
`extends` `Itt_int_base`

### 4.24.2 Terms

`declare`  
`Czf_itt_group_power!power{'g; 'z; 'n}`  
`(displayed as  $\text{power}(g; z; n)$ )`

### 4.24.3 Rewrites

The `power` is defined by induction.

```
![] rewrite unfold_power :
  (power(g; z; n))  $\longleftrightarrow$ 
    (Ind(n) where Ind(n) =
      ((n < 0)
         $\rightarrow$  Ind(n) = (op(g; inv(g; z); power(g; z; n + 1))))
      (n = 0  $\rightarrow$  Ind(n) = id(g))
      ((0 < n)  $\rightarrow$  Ind(n) = (op(g; z; power(g; z; n - 1))))))
```

### 4.24.4 Rules

#### Well-formedness

The `power(g, z, n)` is well-formed if `g` is a label, `z` is a set, and `n` is an integer in ITT.

```
*[3, 371] rule power_wf {| intro [] |} :
  [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash z \text{ set} \longrightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash n \in \mathbb{Z} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (\text{power}(g; z; n)) \text{ set}$ 
```

#### Membership

If `z` is a member of `car(g)`, then `power(g, z, n)` is also in `car(g)` for any integer `n`.

```
*[3, 687] rule power_mem {| intro [] |} :
  [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash n \in \mathbb{Z} \longrightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash z \text{ set} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash z \in s \text{ car}(g) \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (\text{power}(g; z; n)) \in s \text{ car}(g)$ 
```

#### Functionality

The `power` is functional in its set argument.

```
*[13, 2992] rule power_fun {| intro [] |} :
  [.]  $\langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash g \text{ group} \longrightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash n \in \mathbb{Z} \longrightarrow$ 
```

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash \forall z.f[z] \text{ fun\_set} \longrightarrow \\
[ext] \langle \Gamma \rangle; z : \text{set} \vdash f[z] \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.(power(g; f[z]; n)) \text{ fun\_set}
\end{array}$$

### Power reduction

$$x^m * x^n = x^{m+n}$$

```

*[13, 102929] rule power_reduce1 { | intro [] | } :
  [·] ⟨Γ⟩ ⊢ g ∈ Label →
  [ext] ⟨Γ⟩ ⊢ g group →
  [·] ⟨Γ⟩ ⊢ m ∈ ℤ →
  [·] ⟨Γ⟩ ⊢ n ∈ ℤ →
  [·] ⟨Γ⟩ ⊢ x set →
  [ext] ⟨Γ⟩ ⊢ x ∈ s car(g) →
  [ext]
  1. ⟨Γ⟩
  ⊢
  eq((op(g; power(g; x; m); power(g; x; n))); (power(g; x; m + n)))

```

## 4.25 Czf\_itt\_cyclic\_subgroup module

The `Czf_itt_cyclic_subgroup` module defines cyclic subgroups. A cyclic subgroup of group  $g$  generated by  $a$  is a subgroup of  $g$  whose carrier is the collection of  $a^n$  ( $n \in \mathbb{Z}$ ) where  $a \in \text{car}(g)$ . In other words, the carrier is the set of elements in  $\text{car}(g)$  that is equal to  $a^n$  for some  $n \in \mathbb{Z}$ .

### 4.25.1 Parents

```

extends Czf_itt_group_power
extends Czf_itt_subgroup

```

### 4.25.2 Terms

```

declare
  Czf_itt_cyclic_subgroup!cyc_subg{'s; 'g; 'a}
  (displayed as cyclic_subgroup(s; g; a))

```

### 4.25.3 Rewrites

A group  $s$  is a cyclic subgroup of group  $g$  generated by  $a$  if the carrier of  $s$  is the set of elements  $x$  in  $car(g)$  which is equal to  $power(g, a, n)$  for some integer  $n$ , and if the binary operation of  $s$  is the same as that of  $g$ .

```

![] rewrite unfold_cyc_subg :
  cyclic_subgroup(s; g; a)  $\longleftrightarrow$ 
    (s group
      $\wedge$  g group
      $\wedge$  (a  $\in$  s car(g))
      $\wedge$  (car(s) = $'$  { x  $\in$  s car(g) |
                    ( $\exists$ n :  $\mathbb{Z}$ . eq(x; (power(g; a; n)))) } )
      $\wedge$  ( $\forall$ a : set
           $\forall$ b : set
            (a  $\in$  s car(s)
              $\rightarrow$  (b  $\in$  s car(s))
              $\rightarrow$  eq((op(s; a; b)); (op(g; a; b))))))

```

### 4.25.4 Rules

#### Typehood

The  $cyclic\_subgroup(s, g, a)$  is well-formed if  $s$  and  $g$  are labels and  $a$  is a set.

```

*[1, 444] rule cyc_subg_wf {| intro [] |} :
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  g  $\in$  Label  $\longrightarrow$ 
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  s  $\in$  Label  $\longrightarrow$ 
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  a set  $\longrightarrow$ 
  [ext]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  cyclic_subgroup(s; g; a) Type

```

#### Introduction

The proposition  $cyclic\_subgroup(s, g, a)$  is true if it is well-formed,  $s$  and  $g$  are groups,  $a \in_s car(g)$ ,  $equal(car(g), \{x \in_s car(s) \mid \exists n: \mathbb{Z}. eq(x, power(g, a, x))\})$ , and  $op(s, a, b)$  is defined as  $op(g, a, b)$  for  $a, b \in car(s)$ .

```

*[1, 199] rule cyc_subg_intro {| intro [] |} :
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  g  $\in$  Label  $\longrightarrow$ 
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  s  $\in$  Label  $\longrightarrow$ 
  [ext]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  g group  $\longrightarrow$ 
  [ext]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  s group  $\longrightarrow$ 
  [.]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  a set  $\longrightarrow$ 
  [ext]  $\langle$  $\Gamma$  $\rangle$   $\vdash$  a  $\in$  s car(g)  $\longrightarrow$ 
  [ext]
  1.  $\langle$  $\Gamma$  $\rangle$ 

```

$$\begin{aligned} &\vdash \text{car}(s) = \{ x \in s \text{ car}(g) \mid (\exists n : \mathbb{Z}. \text{eq}(x; (\text{power}(g; a; n)))) \} \longrightarrow \\ &[\text{ext}] \\ &1. \langle \Gamma \rangle \\ &2. b : \text{set} \\ &3. c : \text{set} \\ &4. x : b \in s \text{ car}(s) \\ &5. y : c \in s \text{ car}(s) \\ &\vdash \text{eq}((\text{op}(s; b; c)); (\text{op}(g; b; c))) \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_subgroup}(s; g; a) \end{aligned}$$

## Properties

A cyclic subgroup is a subgroup.

\*[3, 554] **rule** `cybsubg_subgroup a` :

$$\begin{aligned} &[\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ &[\cdot] \langle \Gamma \rangle \vdash s \in \text{Label} \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash s \text{ group} \longrightarrow \\ &[\cdot] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_subgroup}(s; g; a) \longrightarrow \\ &[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(s; g) \end{aligned}$$

### 4.25.5 Tactics

`cybsubgSubgroupT` The tactic applies the `cybsubg_subgroup` rule and proves a group is a subgroup by showing it is a cyclic subgroup.

## 4.26 Czf\_itt\_cyclic\_group module

The `Czf_itt_cyclic_group` module defines cyclic groups. A group  $g$  is *cyclic* if there exists  $a \in \text{car}(g)$  such that for every  $x \in \text{car}(g)$  there is an integer  $n$  such that  $\text{eq}(x, a^n)$ .

### 4.26.1 Parents

**extends** `Czf_itt_group`  
**extends** `Czf_itt_cyclic_subgroup`  
**extends** `Czf_itt_abel_group`

## 4.26.2 Terms

```

declare
  Czf_itt_cyclic_group!cycg{'g} (displayed as cyclic_group(g))
declare
  Czf_itt_cyclic_group!cycgroup{'g; 'a}
  (displayed as cyclic_group(g; a))

```

## 4.26.3 Rewrites

A group  $g$  is cyclic if it has a generator. A cyclic group generated by  $a$  can be defined with separation.

```

![] rewrite unfold_cycg :
  cyclic_group(g)  $\longleftrightarrow$ 
    (g group
      $\wedge$  ( $\exists a : set$ 
            $a \in s\ car(g)$ 
            $\wedge$  ( $\forall x : set$ 
                ( $x \in s\ car(g) \rightarrow (\exists n : \mathbb{Z}. eq(x; (power(g; a; n))))$ ))))))
![] rewrite unfold_cycgroup :
  cyclic_group(g; a)  $\longleftrightarrow$ 
    (g group
      $\wedge$  ( $a \in s\ car(g)$ 
            $\wedge$  ( $car(g) = \{ x \in s\ car(g) \mid$ 
                ( $\exists n : \mathbb{Z}. eq(x; (power(g; a; n)))) \}$ )))

```

## 4.26.4 Rules

### Typehood

The  $cyclic\_group(g)$  is well-formed if  $g$  is a label; the  $cyclic\_group(g, a)$  is well-formed if  $g$  is a label and  $a$  is a set.

```

*[1, 323] rule cycg_wf {| intro [] |} :
  [.]  $\langle \Gamma \rangle \vdash g \in Label \rightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash cyclic\_group(g) Type$ 
*[1, 196] rule cycgroup_wf {| intro [] |} :
  [.]  $\langle \Gamma \rangle \vdash g \in Label \rightarrow$ 
  [.]  $\langle \Gamma \rangle \vdash a set \rightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash cyclic\_group(g; a) Type$ 

```

### Functionality

The  $cycgroup$  is functional in its set argument.

\*[17, 18522] **rule cycgroup\_fun**  $\{| \text{intro } [] \}$  :

$$\begin{array}{l} [\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \forall z. \text{cyclic\_group}(g; z) \text{ fun\_prop} \end{array}$$

## Introduction

The proposition  $\text{cyclic\_group}(g)$  is true if it is well-formed, and there is an element  $a$  in its carrier set such that any element in the carrier set is to some power of  $a$ .

The proposition  $\text{cyclic\_group}(g, a)$  is true if it is well-formed,  $a \in_s \text{car}(g)$ , and  $\text{equal}(\text{car}(g), \{x \in_s \text{car}(g) \mid \exists n: \mathbb{Z}. \text{eq}(x, \text{power}(g, a, x))\})$ .

\*[3, 317] **rule cycg\_intro**  $\{| \text{intro } [] \}$   $a$  :

$$\begin{array}{l} [\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ [\cdot] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash a \in_s \text{car}(g) \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle; x : \text{set}; u : x \in_s \text{car}(g) \vdash \exists n : \mathbb{Z}. \text{eq}(x; (\text{power}(g; a; n))) \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_group}(g) \end{array}$$

\*[1, 37] **rule cycgroup\_intro**  $\{| \text{intro } [] \}$  :

$$\begin{array}{l} [\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ [\cdot] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash a \in_s \text{car}(g) \longrightarrow \\ [\text{ext}] \\ 1. \langle \Gamma \rangle \\ \vdash \text{car}(g) = \{ x \in_s \text{car}(g) \mid (\exists n : \mathbb{Z}. \text{eq}(x; (\text{power}(g; a; n)))) \} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_group}(g; a) \end{array}$$

## Theorems

$\text{cyclic\_group}(g)$  is equivalent to there exists  $a \in \text{car}(g)$  such that  $\text{cyclic\_group}(g, a)$ .

\*[6, 1574] **rule cycg1** :

$$\begin{array}{l} [\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_group}(g) \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \exists a \in_s \text{car}(g). \text{cyclic\_group}(g; a) \end{array}$$

\*[4, 1322] **rule cycg2** :

$$\begin{array}{l} [\cdot] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash g \text{ group} \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \exists a \in_s \text{car}(g). \text{cyclic\_group}(g; a) \longrightarrow \\ [\text{ext}] \langle \Gamma \rangle \vdash \text{cyclic\_group}(g) \end{array}$$

## Theorems

Every cyclic group is abelian.

```
*[10, 3532] rule cycg_abel :  
  [·] ⟨Γ⟩ ⊢ g ∈ Label →  
  [ext] ⟨Γ⟩ ⊢ g group →  
  [ext] ⟨Γ⟩ ⊢ cyclic_group(g) →  
  [ext] ⟨Γ⟩ ⊢ abel(g)
```

### 4.26.5 Tactics

`cycgroupAbelT` The tactic applies the `cycgroup_abel` rule and proves a group is abelian by showing it is cyclic.

## 4.27 Czf\_itt\_coset module

The `Czf_itt_coset` module defines the *left coset* and the *right coset*. If  $h$  is a subgroup of  $g$  and  $a \in_s \text{car}(g)$ , then the left coset containing  $a$  is  $\{a * x \mid x \in \text{car}(h)\}$  and the right coset containing  $a$  is  $\{x * a \mid x \in \text{car}(h)\}$ . The elements of the left coset are those in  $\text{car}(g)$  which are equal to  $\text{op}(g, a, y)$  for some  $y \in \text{car}(h)$ . The elements of the right coset are those in  $\text{car}(g)$  which are equal to  $\text{op}(g, y, a)$  for some  $y \in \text{car}(h)$ . The cosets are defined by separation.

### 4.27.1 Parents

```
extends Czf_itt_group  
extends Czf_itt_dexists  
extends Czf_itt_subgroup
```

### 4.27.2 Terms

```
declare  
  Czf_itt_coset!lcoset{'h; 'g; 'a}  
  (displayed as lcoset(h; g; a))  
declare  
  Czf_itt_coset!rcoset{'h; 'g; 'a}  
  (displayed as rcoset(h; g; a))
```

### 4.27.3 Rewrites

The `lcaset` and `rcaset` terms are defined by separation.

```

![] rewrite unfold_lcaset :
  (lcaset(h; g; a))  $\longleftrightarrow$ 
    { x  $\in$  s car(g) | ( $\exists$ y  $\in$  s car(h).eq(x; (op(g; a; y)))) }
![] rewrite unfold_rcaset :
  (rcaset(h; g; a))  $\longleftrightarrow$ 
    { x  $\in$  s car(g) | ( $\exists$ y  $\in$  s car(h).eq(x; (op(g; y; a)))) }

```

### 4.27.4 Rules

#### Well-formedness

The `left_caset(h, g, a)` and `right_caset(h, g, a)` are well-formed if  $h$  and  $g$  are labels, and  $a$  is a set.

```

*[1, 367] rule lcaset_isset { | intro [] | } :
  [·]  $\langle$ Γ $\rangle$   $\vdash$  h  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  g  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  a set  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  (lcaset(h; g; a)) set
*[1, 367] rule rcaset_isset { | intro [] | } :
  [·]  $\langle$ Γ $\rangle$   $\vdash$  h  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  g  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  a set  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  (rcaset(h; g; a)) set

```

#### Introduction

A set  $x$  is a member of `left_caset(h, g, a)` if the left coset is well-formed,  $a \in_s \text{car}(g)$ ,  $x \in_s \text{car}(g)$ , `subgroup(h, g)`, and there exists a set  $y$  such that  $y$  is a member of `car(h)` and  $x$  is equal to `op(g, a, y)` in `car(g)`. The case for `rcaset` is similar.

```

*[1, 556] rule lcaset_intro { | intro [] | } z :
  [·]  $\langle$ Γ $\rangle$   $\vdash$  h  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  g  $\in$  Label  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  a set  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  x set  $\longrightarrow$ 
  [·]  $\langle$ Γ $\rangle$   $\vdash$  z set  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  a  $\in$  s car(g)  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  x  $\in$  s car(g)  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  subgroup(h; g)  $\longrightarrow$ 
  [ext]  $\langle$ Γ $\rangle$   $\vdash$  z  $\in$  s car(h)  $\longrightarrow$ 

```

$$\begin{array}{l}
[ext] \langle \Gamma \rangle \vdash eq(x; (op(g; a; z))) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash x \in_s (lcset(h; g; a)) \\
*[1, 556] \text{ rule } \mathbf{rcset\_intro} \{ | \text{intro } [] | \} z : \\
[.] \langle \Gamma \rangle \vdash h \in Label \longrightarrow \\
[.] \langle \Gamma \rangle \vdash g \in Label \longrightarrow \\
[.] \langle \Gamma \rangle \vdash a \text{ set } \longrightarrow \\
[.] \langle \Gamma \rangle \vdash x \text{ set } \longrightarrow \\
[.] \langle \Gamma \rangle \vdash z \text{ set } \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash a \in_s car(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash x \in_s car(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash subgroup(h; g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash z \in_s car(h) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash eq(x; (op(g; z; a))) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash x \in_s (rcset(h; g; a))
\end{array}$$

### Elimination

The elimination form for the left coset  $y \in_s left\_coset(h, g, a)$  implies  $y \in_s car(g)$  and also produces a witness  $z \in_s car(h)$  for which  $eq(y, op(g, a, z))$ . The case for  $\mathbf{rcset}$  is similar.

$$\begin{array}{l}
*[4, 1839] \text{ rule } \mathbf{lcset\_elim} \{ | \text{elim } [] | \} \Gamma : \\
[.] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash h \in Label \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash g \in Label \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash a \text{ set } \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash y \text{ set } \longrightarrow \\
[ext] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash a \in_s car(g) \longrightarrow \\
[ext] \langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash subgroup(h; g) \longrightarrow \\
[ext] \\
1. \langle \Gamma \rangle \\
2.  $x : y \in_s (lcset(h; g; a))$  \\
3.  $\langle \Delta[x] \rangle$  \\
4.  $u : y \in_s car(g)$  \\
5.  $z : \text{set}$  \\
6.  $v : z \in_s car(h)$  \\
7.  $w : eq(y; (op(g; a; z)))$  \\
 $\vdash C[x] \longrightarrow$  \\
[ext]  $\langle \Gamma \rangle; x : y \in_s (lcset(h; g; a)); \langle \Delta[x] \rangle \vdash C[x]$  \\
*[4, 1839] \text{ rule } \mathbf{rcset\_elim} \{ | \text{elim } [] | \} \Gamma : \\
[.] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash h \in Label \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash g \in Label \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash a \text{ set } \longrightarrow \\
[.] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash y \text{ set } \longrightarrow \\
[ext] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash a \in_s car(g) \longrightarrow \\
[ext] \langle \Gamma \rangle; x : y \in_s (rcset(h; g; a)); \langle \Delta[x] \rangle \vdash subgroup(h; g) \longrightarrow
\end{array}$$

$$\begin{array}{l}
[ext] \\
1. \langle \Gamma \rangle \\
2. x : y \in s \text{ (rcoset}(h; g; a)) \\
3. \langle \Delta[x] \rangle \\
4. u : y \in s \text{ car}(g) \\
5. z : \text{set} \\
6. v : z \in s \text{ car}(h) \\
7. w : eq(y; (op(g; z; a))) \\
\vdash C[x] \longrightarrow \\
[ext] \langle \Gamma \rangle; x : y \in s \text{ (rcoset}(h; g; a)); \langle \Delta[x] \rangle \vdash C[x]
\end{array}$$

## Theorems

If  $h$  is a subgroup of group  $g$ , both the left and right cosets of  $h$  containing  $a$  are subsets of the carrier of  $g$ .

$$\begin{array}{l}
*[1, 848] \text{ rule lcoset\_subset } \{ | \text{intro } [] \} : \\
[ \cdot ] \langle \Gamma \rangle \vdash h \in \text{Label} \longrightarrow \\
[ \cdot ] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ \cdot ] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \text{subgroup}(h; g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash (\text{lcoset}(h; g; a)) \subseteq s \text{ car}(g) \\
*[1, 848] \text{ rule rcoset\_subset } \{ | \text{intro } [] \} : \\
[ \cdot ] \langle \Gamma \rangle \vdash h \in \text{Label} \longrightarrow \\
[ \cdot ] \langle \Gamma \rangle \vdash g \in \text{Label} \longrightarrow \\
[ \cdot ] \langle \Gamma \rangle \vdash a \text{ set} \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash a \in s \text{ car}(g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \text{subgroup}(h; g) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash (\text{rcoset}(h; g; a)) \subseteq s \text{ car}(g)
\end{array}$$

## 4.28 Czf\_itt\_normal\_subgroup module

The `Czf_itt_normal_subgroup` module defines normal subgroups. A subgroup  $h$  of a group  $g$  is *normal* if its left and right cosets coincide, that is, if

$$\forall a \in \text{car}(g). \text{equal}(\text{left\_coset}(h, g, a), \text{right\_coset}(h, g, a))$$

### 4.28.1 Parents

**extends** `Czf_itt_subgroup`  
**extends** `Czf_itt_abel_group`

**extends** Czf\_itt\_coset

## 4.28.2 Terms

**declare**  
 Czf\_itt\_normal\_subgroup!normal\_subg{'s; 'g}  
 (displayed as *normal\_subgroup(s; g)*)

## 4.28.3 Rewrites

A subgroup  $s$  of  $g$  is normal if its left and right cosets coincides.

**!** **rewrite** **unfold\_normal\_subg** :  
 $normal\_subgroup(s; g) \longleftrightarrow$   
 $(subgroup(s; g)$   
 $\wedge (\forall a : set$   
 $(a \in s \ car(g)$   
 $\rightarrow ((lcoset(s; g; a)) =' (rcoset(s; g; a))))))$

## 4.28.4 Rules

### Typehood

The **normal\_subg** judgment is well-formed if its arguments are labels.

**\*[1, 216]** **rule** **normalSubg\_wf** { | intro [] | } :  
 $[\cdot] \langle \Gamma \rangle \vdash s \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash g \in Label \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash normal\_subgroup(s; g) \mathbf{Type}$

### Introduction

The proposition  $normal\_subgroup(s, g)$  is true if it is well-formed,  $s$  is a subgroup of  $g$ , and for any  $a$  in  $car(g)$ ,  $equal(left\_coset(s, g, a), right\_coset(s, g, a))$ .

**\*[1, 88]** **rule** **normalSubg\_intro** { | intro [] | } :  
 $[\cdot] \langle \Gamma \rangle \vdash s \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle \vdash g \in Label \longrightarrow$   
 $[ext] \langle \Gamma \rangle \vdash subgroup(s; g) \longrightarrow$   
 $[ext]$   
 1.  $\langle \Gamma \rangle$   
 2.  $a : set$   
 3.  $x : a \in s \ car(g)$   
 $\vdash (lcoset(s; g; a)) =' (rcoset(s; g; a)) \longrightarrow$

$[ext] \langle \Gamma \rangle \vdash normal\_subgroup(s; g)$

## Theorems

All subgroups of abelian groups are normal.

\*[10, 1803] **rule** `abel_subg_normal`  $\Gamma$   $s$  :  
 $[\cdot] \langle \Gamma \rangle; x : abel(g); \langle \Delta[x] \rangle \vdash s \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle; x : abel(g); \langle \Delta[x] \rangle \vdash g \in Label \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : abel(g); \langle \Delta[x] \rangle \vdash subgroup(s; g) \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : abel(g); \langle \Delta[x] \rangle; y : normal\_subgroup(s; g) \vdash C[x] \longrightarrow$   
 $[ext] \langle \Gamma \rangle; x : abel(g); \langle \Delta[x] \rangle \vdash C[x]$

## 4.29 Czf\_itt\_inv\_image module

The `Czf_itt_inv_image` module defines the *inverse image* of a set under some mapping. The inverse image is defined as a set constructor  $\{x \in_s s \mid t \in_s a[x]\}$ . The term  $s$  and  $t$  must be sets, and  $a[x]$  must be functional. The elements of the inverse image are the elements of  $x$  in  $s$  for which  $a[x]$  in  $t$  is true.

### 4.29.1 Parents

`extends` `Czf_itt_sep`

### 4.29.2 Terms

**declare**  
`Czf_itt_inv_image!``inv_image`{'s; x. a['x]; 't}  
 (displayed as  $\{x \in_s s \mid a[x] \in_s t\}$ )

### 4.29.3 Rewrites

The `inv_image` term is defined by separation.

**![]** **rewrite** `unfold_inv_image` :  
 $(\{x \in_s s \mid a[x] \in_s t\}) \longleftrightarrow \{x \in_s s \mid (a[x] \in_s t)\}$

## 4.29.4 Rules

### Well-formedness

The inverse image  $\{x \in_s s \mid a[x] \in_s t\}$  is well-formed if  $s$  and  $t$  are sets, and  $a[x]$  is functional.

$$\begin{array}{l}
 * [2, 90] \text{ rule } \mathit{inv\_image\_isset} \{ | \text{intro } [] \} : \\
 [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow \\
 [\cdot] \langle \Gamma \rangle \vdash t \text{ set} \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash \forall z. a[z] \text{ fun\_set} \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash (\{x \in_s s \mid a[x] \in_s t\}) \text{ set}
 \end{array}$$

### Introduction

A set  $y$  is a member of  $\{x \in_s s \mid a[x] \in_s t\}$  if the inverse image is well-formed; if  $y \in_s s$ ; and if  $a[y] \in_s t$ .

$$\begin{array}{l}
 * [1, 114] \text{ rule } \mathit{inv\_image\_intro} \{ | \text{intro } [] \} : \\
 [\cdot] \langle \Gamma \rangle \vdash y \text{ set} \longrightarrow \\
 [\cdot] \langle \Gamma \rangle \vdash s \text{ set} \longrightarrow \\
 [\cdot] \langle \Gamma \rangle \vdash t \text{ set} \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash \forall x. a[x] \text{ fun\_set} \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash y \in_s s \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash a[y] \in_s t \longrightarrow \\
 [ext] \langle \Gamma \rangle \vdash y \in_s (\{x \in_s s \mid a[x] \in_s t\})
 \end{array}$$

### Elimination

An assumption  $y \in_s \{x \in_s s \mid a[x] \in_s t\}$  implies two facts:  $y \in_s s$  and  $a[y] \in_s t$ .

$$\begin{array}{l}
 * [1, 149] \text{ rule } \mathit{inv\_image\_elim} \{ | \text{elim } [] \} \Gamma : \\
 [\cdot] \langle \Gamma \rangle; x : y \in_s (\{x \in_s s \mid a[x] \in_s t\}); \langle \Delta[x] \rangle \vdash y \text{ set} \longrightarrow \\
 [\cdot] \langle \Gamma \rangle; x : y \in_s (\{x \in_s s \mid a[x] \in_s t\}); \langle \Delta[x] \rangle \vdash s \text{ set} \longrightarrow \\
 [\cdot] \langle \Gamma \rangle; x : y \in_s (\{x \in_s s \mid a[x] \in_s t\}); \langle \Delta[x] \rangle \vdash t \text{ set} \longrightarrow \\
 [ext] \langle \Gamma \rangle; x : y \in_s (\{x \in_s s \mid a[x] \in_s t\}); \langle \Delta[x] \rangle \vdash \forall x. a[x] \text{ fun\_set} \longrightarrow \\
 [ext] \\
 1. \langle \Gamma \rangle \\
 2.  $x : y \in_s (\{x \in_s s \mid a[x] \in_s t\})$  \\
 3.  $\langle \Delta[x] \rangle$  \\
 4.  $v : y \in_s s$  \\
 5.  $w : a[y] \in_s t$  \\
  $\vdash C[x] \longrightarrow$  \\
 [ext]  $\langle \Gamma \rangle; x : y \in_s (\{x \in_s s \mid a[x] \in_s t\}); \langle \Delta[x] \rangle \vdash C[x]$ 
 \end{array}$$

## 4.30 Czf\_itt\_hom module

The `Czf_itt_hom` module defines the *homomorphism*. A homomorphism is a mapping  $f$  from one group  $g_1$  into another group  $g_2$ , which satisfies for any  $a$  and  $b$  in  $car(g_1)$ ,

$$f(a*_1b) = f(a)*_2f(b)$$

$f$  is a mapping from group  $g_1$  into group  $g_2$  means: first, for any  $a$  in  $car(g_1)$ ,  $f(a)$  is in  $car(g_2)$ ; second, for each  $a$  in  $car(g_1)$ , *exactly* one element is assigned in  $car(g_2)$ .

The homomorphism is defined as follows:

$$\begin{aligned} \text{hom}(f[x] : g_1 - > g_2) &\equiv \\ &\text{group}(g_1) \\ &\wedge \text{group}(g_2) \\ &\wedge \forall a: \text{set.}(a \in_s \text{car}(g_1) \Rightarrow f[a] \in_s \text{car}(g_2)) \\ &\wedge \forall a: \text{set.}\forall b: \text{set.}(a \in_s \text{car}(g_1) \Rightarrow b \in_s \text{car}(g_1) \\ &\quad \Rightarrow \text{eq}(a, b) \Rightarrow \text{eq}(f[a], f[b])) \\ &\wedge \forall a: \text{set.}\forall b: \text{set.}(a \in_s \text{car}(g_1) \Rightarrow b \in_s \text{car}(g_1) \\ &\quad \Rightarrow \text{eq}(f[\text{op}(g_1, a, b)], \text{op}(g_2, f[a], f[b]))) \end{aligned}$$

### 4.30.1 Parents

```

extends Czf_itt_group
extends Czf_itt_subgroup
extends Czf_itt_abel_group
extends Czf_itt_inv_image

```

### 4.30.2 Terms

```

declare
  Czf_itt_hom!hom{'g1; 'g2; x. f['x]}
  (displayed as hom(g1; g2; f[x]))

```

### 4.30.3 Rewrites

The `hom` judgment requires that  $g_1$  and  $g_2$  be groups,  $f$  be a mapping from  $car(g_1)$  into  $car(g_2)$ , and for any  $a$  and  $b$  in  $car(g_1)$ ,  $f$  map  $op(g_1, a, b)$  into  $op(g_2, f[a], f[b])$ .

```

[!] rewrite unfold_hom :
  (hom(g1; g2; f[x]))  $\longleftrightarrow$ 
  (g1_group
    $\wedge$  g2_group)

```

$$\begin{aligned}
& \wedge (\forall a : \text{set}. (a \in s \text{ car}(g_1) \rightarrow (f[a] \in S \text{ car}(g_2)))) \\
& \wedge (\forall a : \text{set} \\
& \quad \forall b : \text{set} \\
& \quad \quad (a \in s \text{ car}(g_1) \\
& \quad \quad \rightarrow (b \in s \text{ car}(g_1)) \\
& \quad \quad \rightarrow \text{eq}(a; b) \\
& \quad \quad \rightarrow \text{eq}(f[a]; f[b]))) \\
& \wedge (\forall a : \text{set} \\
& \quad \forall b : \text{set} \\
& \quad \quad (a \in s \text{ car}(g_1) \\
& \quad \quad \rightarrow (b \in s \text{ car}(g_1)) \\
& \quad \quad \rightarrow \text{eq}(f[(\text{op}(g_1; a; b))]; (\text{op}(g_2; f[a]; f[b])))))
\end{aligned}$$

#### 4.30.4 Rules

##### Well-formedness

The `hom` is well-formed if  $g_1$  and  $g_2$  are labels, and  $f[x]$  is a set for any set argument  $x$ .

$$\begin{aligned}
& * [3, 852] \text{ rule } \text{hom\_type} \{ | \text{ intro } [] | \} : \\
& \quad [\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
& \quad [\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
& \quad [\cdot] \langle \Gamma \rangle; x : \text{set} \vdash f[x] \text{set} \longrightarrow \\
& \quad [\text{ext}] \langle \Gamma \rangle \vdash (\text{hom}(g_1; g_2; f[x])) \text{Type}
\end{aligned}$$

##### Introduction

The proposition  $\text{hom}(f[x] : g_1 \rightarrow g_2)$  is true if it is well-formed,  $g_1$  and  $g_2$  are groups,  $f$  assigns to each element  $x$  of  $\text{car}(g_1)$  exactly one element  $b$  of  $\text{car}(g_2)$ , and  $f$  maps  $\text{op}(g_1, a, b)$  into  $\text{op}(g_2, f[a], f[b])$  for any  $a$  and  $b$  in  $\text{car}(g_1)$ .

$$\begin{aligned}
& * [1, 430] \text{ rule } \text{hom\_intro} \{ | \text{ intro } [] | \} : \\
& \quad [\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
& \quad [\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
& \quad [\text{ext}] \langle \Gamma \rangle \vdash g_1 \text{ group} \longrightarrow \\
& \quad [\text{ext}] \langle \Gamma \rangle \vdash g_2 \text{ group} \longrightarrow \\
& \quad [\text{ext}] \langle \Gamma \rangle; x : \text{set}; y : x \in s \text{ car}(g_1) \vdash f[x] \in S \text{ car}(g_2) \longrightarrow \\
& \quad [\text{ext}] \\
& \quad 1. \langle \Gamma \rangle \\
& \quad 2. c : \text{set} \\
& \quad 3. d : \text{set} \\
& \quad 4. x_1 : c \in s \text{ car}(g_1) \\
& \quad 5. y_1 : d \in s \text{ car}(g_1) \\
& \quad 6. u : \text{eq}(c; d)
\end{aligned}$$

$$\begin{array}{l}
\vdash eq(f[c]; f[d]) \longrightarrow \\
[ext] \\
1. \langle \Gamma \rangle \\
2. e : set \\
3. g : set \\
4. x_2 : e \in s\ car(g_1) \\
5. y_2 : g \in s\ car(g_1) \\
\vdash eq(f[(op(g_1; e; g))]; (op(g_2; f[e]; f[g]))) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash hom(g_1; g_2; f[x])
\end{array}$$

### Functionality

The `hom` judgment is functional in the function argument.

$$\begin{array}{l}
*[23, 4266] \text{ rule } hom\_fun \{ | \text{ intro } [] \} : \\
[\cdot] \langle \Gamma \rangle \vdash g_1 \in Label \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash g_2 \in Label \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g_1\ group \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g_2\ group \longrightarrow \\
[ext] \langle \Gamma \rangle; z : set; x_1 : set; y_1 : x_1 \in s\ car(g_1) \vdash f[z; x_1] \in s\ car(g_2) \longrightarrow \\
[ext] \langle \Gamma \rangle; z : set \vdash \forall x.f[x; z]\ fun\_set \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash \forall z.(hom(g_1; g_2; f[z; y]))\ fun\_prop
\end{array}$$

### Trivial homomorphism

For any groups  $g_1$  and  $g_2$ , there is always at least one homomorphism  $f: g_1 \rightarrow g_2$  which maps all elements of  $car(g_1)$  into  $id(g_2)$ . This is called the trivial homomorphism.

$$\begin{array}{l}
*[11, 2680] \text{ rule } trivial\_hom1 : \\
[\cdot] \langle \Gamma \rangle \vdash g_1 \in Label \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash g_2 \in Label \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g_1\ group \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash g_2\ group \longrightarrow \\
[ext] \langle \Gamma \rangle; x : set; y : x \in s\ car(g_1) \vdash f[x] = ' id(g_2) \longrightarrow \\
[ext] \langle \Gamma \rangle \vdash hom(g_1; g_2; f[x])
\end{array}$$

### Theorems

Let  $f: g_1 \rightarrow g_2$  be a group homomorphism of  $g_1$  into  $g_2$ .

If  $f$  is *onto*, then  $g_1$  is abelian implies  $g_2$  is abelian.

$$\begin{array}{l}
*[17, 7055] \text{ rule } hom\_abel (hom(g_1; g_2; f[x])) : \\
[\cdot] \langle \Gamma \rangle \vdash g_1 \in Label \longrightarrow
\end{array}$$

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. x : \text{set} \\
3. y : x \in s \text{ car}(g_2) \\
\vdash \exists z : \text{set}. z \in s \text{ car}(g_1) \wedge \text{eq}(x; f[z]) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{hom}(g_1; g_2; f[x]) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{abel}(g_1) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{abel}(g_2)
\end{array}$$

$f$  maps the identity of  $g_1$  into the identity of  $g_2$ .

\*[9, 5122] **rule hom\_id\_elim**  $\Gamma$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \text{hom}(g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \text{eq}(f[\text{id}(g_1)]; \text{id}(g_2)) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

$f$  maps the inverse of an element  $a$  in  $\text{car}(g_1)$  into the inverse of  $f[a]$  in  $\text{car}(g_2)$ .

\*[19, 6440] **rule hom\_inv\_elim**  $\Gamma a$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{hom}(g_1; g_2; f[x]) \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \text{ set} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \in s \text{ car}(g_1) \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \text{hom}(g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \text{eq}(f[(\text{inv}(g_1; a))]; (\text{inv}(g_2; f[a]))) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \text{hom}(g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

If  $h$  is a subgroup of  $g_1$ , then the image of  $h$  under  $f$  is a subgroup of  $g_2$ .

\*[4, 910] **rule hom\_subg1** ( $\text{hom}(g_1; g_2; f[x])$ )  $h_1 h_2$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow
\end{array}$$

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash h_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h_2 \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{hom}(g_1; g_2; f[x]) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_1; g_1) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash h_2 \text{ group} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
\vdash \text{car}(h_2) = ' \{ x \in s \text{ car}(g_2) \mid (\exists y \in s \text{ car}(h_1)).\text{eq}(x; f[y]) \} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. a : \text{set} \\
3. b : \text{set} \\
4. x : a \in s \text{ car}(h_2) \\
5. y : b \in s \text{ car}(h_2) \\
\vdash \text{eq}((\text{op}(h_2; a; b)); (\text{op}(g_2; a; b))) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_2; g_2)
\end{array}$$

If  $k$  is a subgroup of  $g_2$ , then the inverse image of  $k$  under  $f$  is a subgroup of  $g_1$ .

\*[2, 702] **rule hom\_subg2** ( $\text{hom}(g_1; g_2; f[x])$ )  $h_1 h_2$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle \vdash h_2 \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{hom}(g_1; g_2; f[x]) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_2; g_2) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash h_1 \text{ group} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{car}(h_1) = ' (\{x \in s \text{ car}(g_1) \mid f[x] \in s \text{ car}(h_2)\}) \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. a : \text{set} \\
3. b : \text{set} \\
4. x : a \in s \text{ car}(h_1) \\
5. y : b \in s \text{ car}(h_1) \\
\vdash \text{eq}((\text{op}(h_1; a; b)); (\text{op}(g_1; a; b))) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle \vdash \text{subgroup}(h_1; g_1)
\end{array}$$

### 4.30.5 Tactics

**homIdT**, **homInvT** The **homIdT** applies the **hom\_id\_elim** rule, and the **homInvT** tactic applies the **hom\_inv\_elim** rule. They infer the mapping relations of the identity and inverse between two groups under a homomorphism.

## 4.31 Czf\_itt\_iso module

The `Czf_itt_iso` module defines the isomorphism.

### 4.31.1 Parents

```
extends Czf_itt_group
extends Czf_itt_hom
```

### 4.31.2 Terms

```
declare
  Czf_itt_iso!iso{'g1; 'g2; x. f['x]}
  (displayed as iso(g1; g2; f[x]))
```

### 4.31.3 Rewrites

An isomorphism is a homomorphism that is one-to-one and onto G2.

```
![] rewrite unfold_iso :
  (iso(g1; g2; f[x]))  $\longleftrightarrow$ 
  ((hom(g1; g2; f[x]))
    $\wedge$  ( $\forall c : set$ 
         $\forall d : set$ 
          (c  $\in$  s car(g1)
            $\rightarrow$  (d  $\in$  s car(g1))
                    $\rightarrow$  eq(f[c]; f[d])
                    $\rightarrow$  eq(c; d))))
    $\wedge$  ( $\forall e : set$ 
        (e  $\in$  s car(g2)  $\rightarrow$  ( $\exists p : set. p \in s car(g1) \wedge eq(e; f[p]))))$ ))
```

### 4.31.4 Rules

#### Well-formedness

The proposition  $iso(f[x] : g1 \rightarrow g2)$  is well-formed if  $g1$  and  $g2$  are labels, and  $f[x]$  is a set for any set argument  $x$ .

```
*[1, 429] rule iso_type { | intro [] | } :
  [·]  $\langle \Gamma \rangle \vdash g1 \in Label \longrightarrow$ 
  [·]  $\langle \Gamma \rangle \vdash g2 \in Label \longrightarrow$ 
  [·]  $\langle \Gamma \rangle; x : set \vdash f[x] set \longrightarrow$ 
  [ext]  $\langle \Gamma \rangle \vdash (iso(g1; g2; f[x])) Type$ 
```

## Functionality

The `iso` judgment is functional in the function argument.

```
*[16, 2490] rule iso_fun { | intro [] | } :
  [·] ⟨Γ⟩ ⊢ g1 ∈ Label →
  [·] ⟨Γ⟩ ⊢ g2 ∈ Label →
  [ext] ⟨Γ⟩ ⊢ g1 group →
  [ext] ⟨Γ⟩ ⊢ g2 group →
  [ext] ⟨Γ⟩; z : set; x1 : set; y1 : x1 ∈ s car(g1) ⊢ f[z; x1] ∈ s car(g2) →
  [ext] ⟨Γ⟩; z : set ⊢ ∀x.f[x; z] fun_set →
  [ext] ⟨Γ⟩ ⊢ ∀z.(iso(g1; g2; f[z; y])) fun_prop
```

## 4.32 Czf\_itt\_ker module

The `Czf_itt_ker` module defines the kernel proposition  $ker(h, g_1, g_2, x.f[x])$ , in which  $f$  is a homomorphism of  $g_1$  into  $g_2$ , i.e.,  $hom(f : g_1 \rightarrow g_2)$ , and  $h$  is a group formed by the elements of  $g_1$  that are mapped into the identity of  $g_2$ .

### 4.32.1 Parents

```
extends Czf_itt_hom
extends Czf_itt_coset
extends Czf_itt_normal_subgroup
```

### 4.32.2 Terms

```
declare
  Czf_itt_ker!ker{ 'h; 'g1; 'g2; x. f['x] }
  (displayed as ker(h; g1; g2; f[x]))
```

### 4.32.3 Rewrites

The `ker` judgment requires that  $hom(f[x] : g_1 \rightarrow g_2)$  and  $h$  be a group which has the same binary operation as  $g_1$  and the elements of whose carrier are all mapped into the identity of  $g_2$ .

```
![] rewrite unfold_ker :
  (ker(h; g1; g2; f[x])) ↔
  ((hom(g1; g2; f[x]))
   ∧ h_group
   ∧ (car(h) = { x ∈ s car(g1) | eq(f[x]; id(g2)) })
```

$$\begin{aligned} & \wedge (\forall a : \text{set} \\ & \quad \forall b : \text{set} \\ & \quad (a \in s \text{ car}(h) \\ & \quad \rightarrow (b \in s \text{ car}(h)) \\ & \quad \rightarrow \text{eq}((\text{op}(h; a; b)); (\text{op}(g_1; a; b)))))) \end{aligned}$$

### 4.32.4 Rules

#### Well-formedness

The kernel proposition  $\text{ker}(h, g_1, g_2, x.f[x])$  is well-formed if  $g_1$ ,  $g_2$ , and  $h$  are labels, and  $f[x]$  is functional in any set argument  $x$ .

$$\begin{aligned} & * [1, 470] \text{ rule ker.type } \{ | \text{intro } [] | \} : \\ & [\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\ & [\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\ & [\cdot] \langle \Gamma \rangle \vdash h \in \text{Label} \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash \forall x.f[x] \text{ fun.set} \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash (\text{ker}(h; g_1; g_2; f[x])) \text{ Type} \end{aligned}$$

#### Introduction

The proposition  $\text{ker}(h, g_1, g_2, x.f[x])$  is true if  $\text{hom}(f : g_1 \rightarrow g_2)$  is true and  $h$  is a group formed by the elements of group  $g_1$  that are mapped into  $\text{id}(g_2)$ .

$$\begin{aligned} & * [1, 185] \text{ rule ker.intro } \{ | \text{intro } [] | \} : \\ & [\cdot] \langle \Gamma \rangle \vdash g_1 \in \text{Label} \longrightarrow \\ & [\cdot] \langle \Gamma \rangle \vdash g_2 \in \text{Label} \longrightarrow \\ & [\cdot] \langle \Gamma \rangle \vdash h \in \text{Label} \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash \text{hom}(g_1; g_2; f[x]) \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash h \text{ group} \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash \text{car}(h) = \{ x \in s \text{ car}(g_1) \mid \text{eq}(f[x]; \text{id}(g_2)) \} \longrightarrow \\ & [\text{ext}] \\ & 1. \langle \Gamma \rangle \\ & 2. a : \text{set} \\ & 3. b : \text{set} \\ & 4. x : a \in s \text{ car}(h) \\ & 5. y : b \in s \text{ car}(h) \\ & \vdash \text{eq}((\text{op}(h; a; b)); (\text{op}(g_1; a; b))) \longrightarrow \\ & [\text{ext}] \langle \Gamma \rangle \vdash \text{ker}(h; g_1; g_2; f[x]) \end{aligned}$$

#### Theorems

The kernel of a group homomorphism from  $g_1$  into  $g_2$  is a subgroup of  $g_1$ .

\*[3, 676] **rule ker\_subgroup\_elim**  $\Gamma$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \ker(h; g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \text{subgroup}(h; g_1) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

If the proposition  $\ker(h, g_1, g_2, x.f[x])$  is true, then the set  $\{x \in_s \text{car}(g_1) \mid \text{eq}(f[x], f[a])\}$  is equal to  $\text{left\_coset}(h, g_1, a)$  and  $\text{right\_coset}(h, g_1, a)$ .

\*[47, 29280] **rule ker\_lcoset\_e**  $\Gamma \ g_2 \ a$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \text{ set} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \in s \text{ car}(g_1) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \ker(h; g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \{x \in s \text{ car}(g_1) \mid \text{eq}(f[x]; f[a])\} = ' (\text{lcoset}(h; g_1; a)) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

\*[47, 28481] **rule ker\_rcoset\_e**  $\Gamma \ g_2 \ a$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \text{ set} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash a \in s \text{ car}(g_1) \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \ker(h; g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \{x \in s \text{ car}(g_1) \mid \text{eq}(f[x]; f[a])\} = ' (\text{rcoset}(h; g_1; a)) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

A group homomorphism  $f$  from  $g_1$  into  $g_2$  is called a *monomorphism* if it is *one to one*; this is the case if and only if the kernel of  $f$  equals  $\{id(g_1)\}$ .

\*[18, 3432] **rule ker\_mono1**  $\Gamma$ :

$[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in Label \longrightarrow$   
 $[ext] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] fun\_set \longrightarrow$   
 $[ext]$

1.  $\langle \Gamma \rangle$

2.  $u : ker(h; g_1; g_2; f[x])$

3.  $\langle \Delta[u] \rangle$

$\vdash car(h) = \{ x \in s car(g_1) \mid eq(x; id(g_1)) \} \longrightarrow$

$[ext]$

1.  $\langle \Gamma \rangle$

2.  $u : ker(h; g_1; g_2; f[x])$

3.  $\langle \Delta[u] \rangle$

$\vdash$

$\forall c : set$

$\forall d : set$

$(c \in s car(g_1) \rightarrow (d \in s car(g_1)) \rightarrow eq(f[c]; f[d]) \rightarrow eq(c; d))$

\*[19, 5823] **rule ker\_mono2**  $\Gamma$ :

$[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in Label \longrightarrow$   
 $[\cdot] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in Label \longrightarrow$   
 $[ext] \langle \Gamma \rangle ; u : ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] fun\_set \longrightarrow$   
 $[ext]$

1.  $\langle \Gamma \rangle$

2.  $u : ker(h; g_1; g_2; f[x])$

3.  $\langle \Delta[u] \rangle$

4.  $c : set$

5.  $d : set$

6.  $v : c \in s car(g_1)$

7.  $w : d \in s car(g_1)$

8.  $z : eq(f[c]; f[d])$

$\vdash eq(c; d) \longrightarrow$

$[ext]$

1.  $\langle \Gamma \rangle$

2.  $u : ker(h; g_1; g_2; f[x])$

3.  $\langle \Delta[u] \rangle$

$\vdash car(h) = \{ x \in s car(g_1) \mid eq(x; id(g_1)) \}$

The kernel of a group homomorphism  $f$  from  $g_1$  into  $g_2$  is a normal subgroup of  $g_1$ .

\*[6, 675] **rule ker\_normalSubg**  $\Gamma$ :

$$\begin{array}{l}
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_1 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash g_2 \in \text{Label} \longrightarrow \\
[\cdot] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash h \in \text{Label} \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash \forall x.f[x] \text{ fun\_set} \longrightarrow \\
[\text{ext}] \\
1. \langle \Gamma \rangle \\
2. u : \ker(h; g_1; g_2; f[x]) \\
3. \langle \Delta[u] \rangle \\
4. v : \text{normal\_subgroup}(h; g_1) \\
\vdash C[u] \longrightarrow \\
[\text{ext}] \langle \Gamma \rangle; u : \ker(h; g_1; g_2; f[x]); \langle \Delta[u] \rangle \vdash C[u]
\end{array}$$

### 4.32.5 Tactics

`kerSubgT`, `kerLcosetT`, `kerRcosetT`, `kerNormalSubgT` The four `ker` tactics apply the theorems for the `ker` judgment. The `kerSubgT` applies the `ker_subgroup_elim` rule, the `kerLcosetT` tactic applies the `ker_lcoset2` rule, the `kerRcosetT` tactic applies the `ker_rcoset2` rule, and the `kerNormalSubgT` tactic applies the `ker_normalSubg` rule.

## Chapter 5

# Mojave FIR Theory

The Mojave FIR Theory (hereafter the FIR theory) seeks to formalize the “Function Intermediate Representation” (FIR) of the Mojave compiler. The FIR is an explicitly typed language that supports polymorphism, process migration, and transactions [HSA<sup>+</sup>02].

The FIR theory differs from the Mojave Compiler theory in its approach to modeling the FIR. Here, FIR programs are represented using sequents, and the results of [HSA<sup>+</sup>02] have been used as a basis for developing the MetaPRL formalization.

Our current goal is to formalize the FIR type system, and then use that formalization to type-check programs from the Mojave compiler.

### 5.1 Mfir\_theory module

The `Mfir_theory` module collects all the modules of the FIR theory.

```
extends Base_theory
extends Mfir_option
extends Mfir_bool
extends Mfir_token
extends Mfir_record
extends Mfir_int
extends Mfir_list
extends Mfir_int_set
extends Mfir_ty
extends Mfir_exp
extends Mfir_util
extends Mfir_sequent
extends Mfir_tr_base
extends Mfir_tr_types
```