A documentation for the accompanying Agda code

Version of January 30, 2026

1 Introduction

To show axioms implies box relations involves a large amount of computation, which is error-prone if we compute them by hand. Computing them in ordinary (non-proof-assistant) software might help, but it hard to argue that the computation is mathematically correct. So we use proof assitant, and the particular choice is Agda [1].

In Agda, we can define sets and relations on a set, for example:

```
data X : Set where
  gen1 : X
  gen2 : X
  gen3 : X
```
(1)

defines a set with three element gen1, gen2 and gen3. And

```
data _===_ : X → X → Set where
  eq : gen1 === gen2
```
(2)

defines a relation on X which relates two elements. Here we call eq an axiom which wittness the relation of gen1 and gen2. By adding more axioms, we can get an equivalence relation on X (the subscript $_1$ is to avoid name clashing):

```
data _===₁_ : X → X → Set where
  eq : gen1 ===₁ gen2
  refl : ∀ {x : X} → x ===₁ x
  sym : ∀ {x y : X} → x ===₁ y → y ===₁ x
  trans : ∀ {x y z : X} → x ===₁ y → y ===₁ z → x ===₁ z
```
(3)

The the above code defines a transitive, symmetric, and reflexive closure of the axiom eq. The Agda standard library [2] provides a way to reasoning equivalence relations called setoid reasoning (a setoid is a set together with an equivalence relation on it). For example, we can show gen2 is related to gen1 by:

```
gen2-gen1 : gen2 ===₁ gen1
gen2-gen1 = begin
  gen2 ≅⟨ sym eq ⟩
  gen1  □
```

which proves gen2 is related to gen1 by the symmetry of axiom eq. Agda checks if the relation claimed is indeed derivable from axioms. If it fails, Agda reports an error.

2 Word setoid

Normally, we model a circuit as a list of gates, and circuit composition as list concatenation, but list concatenation loses information of its operands, i.e., you cannot recover lhs and rhs from lhs ++ rhs in Agda. We model circuit as words of gate set. A word is just a binary tree, where two branches keep track of the two operands of concatenation:

```
data Word (X : Set) : Set where
  [_]ʷ : X → Word X
  ϵ : Word X
  _•_ : Word X → Word X → Word X
```

Here X is any gate set. The leaf node is either a gate or an empty circuit. Circuit equivalence is modelled by an equivlence relation on words that also respects the operation •. Such relation is called a congruence relation. To be precsie, the congruence closure of the a given relation Γ is

```
module Congruence-Closure (Γ : Word X → Word X → Set) where
  infix 4 _≅_
  data _≅_ : Word X → Word X → Set where
    -- ≅ is a congruence.
    refl : w ≅ w
    sym : w ≅ v → v ≅ w
    trans : w ≅ v → v ≅ u → w ≅ u
    cong : w ≅ w' → v ≅ v' → w • v ≅ w' • v'

    -- The monoid axioms: asociativity and the left and right unit laws.
    assoc : (w • v) • u ≅ w • (v • u)
    left-unit : ϵ • w ≅ w
    right-unit : w • ϵ ≅ w

    -- Axioms.
    axiom : Γ w v → w ≅ v
```

Note that axioms of asociativity and the left and right unit laws is to make the concatenation of words compatible the usual concatenation of lists, which also make the word setoid a monoid.

3 Single-ququint example

We use gate set:

```
data Gate-Set : Set where
  S-gen : Gate-Set
  H-gen : Gate-Set
```

We lift gate to circuit by:

```
H : Word Gate-Set
H = [ H-gen ]ʷ
```

```
S : Word Gate-Set
S = [ S-gen ]ʷ
```

We claim the following axiom is complete for the single-ququint symplectic group:

```
data _===₅_ : Word Gate-Set → Word Gate-Set → Set where
  order-S : S ^ 5 ===₅ ϵ
  order-H : H ^ 2 ===₅ M₂ ^ 2
  M-power : ∀ (k : ℕ) → (k ¡ 5) → M₂ ^ k ===₅ M (₂^ k)
  semi-MS : M₂ • S ===₅ S^ (₂ * ₂) • M₂
```

That is the congruence clouse of it implies all single-ququint box relations. The operators $*$ and ${}_2\wedge$ are multiplication and powers of base 2 in $\mathbb{Z}_5$, and

$$Mx = S^x \bullet H \bullet S^{\frac{1}{x}} \bullet H \bullet S^x \bullet H$$

We show one of the box relation follows from these axioms:

```
lemma-single-qupit-br-E : ∀ (b : ℤ ₅) →

  [ b ]ᵉ • S ≅ [ b + - ₁ ]ᵉ

lemma-single-qupit-br-E b = begin
  [ b ]ᵉ • S ≅⟨ refl ⟩
  S^ (- b) • S ≅⟨ lemma-S[k+l] (- b) ₁ ⟩
  S^ (- b + ₁) ≡⟨ Eq.cong S^ (Eq.cong (- b +_) (Eq.sym (-involutive ₁))) ⟩
  S^ (- b + - - ₁) ≡⟨ Eq.cong S^ (-+-comm b (- ₁)) ⟩
  S^ (- (b + - ₁)) ≅⟨ refl ⟩
  [ b + - ₁ ]ᵉ   □
```

Here [b]ᵉ is the E box with parameter b. The proof involves calling a lemma lemma-S[k+l], arithmetics in $\mathbb{Z}_5$ (such as -involutive meaning taking inverse is involutive), and reflexitivity.

4 Derivation of box relations

The statement that box relations are derivable from axioms is in "./BoxRelations.agda", which also contains proof pointers.

The proofs are located at the folders "./N/BR/One/", "./N/BR/Two/", and "./N/BR/Three/" for one, two, and three qupit box relations. In each folder, file name starts with letters "A B D E L" indicating the file is about the corresponding box. For example, in folder "./N/BR/Two/", the file "L-CZ" contains the proof of the rewrite rules of pushing a CZ gate through a 2-qupit L box is derivable.

5 The simplified relations

The proofs in BR folder is actually based on a more "comprehensive" set of axioms, instead of the axioms in Fig 1. We show two sets of axioms are equivalent by showing there is a group isomorphism between two word setoids. The isomorphism can be found in "./N/Symplectic-Simplified.agda" file; axioms in Fig 1 is also encoded here. The more "comprehensive" axioms can be found in "./N/Symplectic.agda".

6 Agda and standard library version

Agda version 2.8.0 and standard library version 2.3 are used.

References

[1] Agda documentation. `https://agda.readthedocs.io/`. Accessed: 2026-01-15.

[2] Agda standard library. `https://agda.github.io/agda-stdlib/v2.3/`. Accessed: 2026-1-15.