有限巡回群の作用をもつ多様体

(課題番号：10640068)

平成10年度～平成11年度科学研究費補助金 (基盤研究 (C)(2)) 研究成果報告書

平成12年3月

研究代表者　北田　泰彦
(横浜国立大学工学部教授)

はしがき

有限巡回群の作用を持つ多様体として、最初に考えるべき多様体は球面であり、1970年以降手術理論を適用して研究されて来た。作用の存在および分類の問題を扱うときには、およそ2つの方法がある。その一つは手術障害群に着目する代数的なものであり、もう一つは法コボルディズム類を扱うホモトピー的な方法がある。位相多様体あるいはPL多様体では、手術の分類空間 F/TOP や F/PL の構造が簡単であるため、法コボルディズム類を扱うことは容易で、手術障害群のみを考えることでこと足りる。一方微分可能なカテゴリーでは分類空間 F/O の構造は本質的に球面の安定ホモトピー群と同じ複雑さを持っており、法コボルディズム類を決定することはほとんど不可能である。しかし、複雑な法コボルディズム類全体を知ることは無理としても、法コボルディズム類の持つ情報ののうちで幾何学的に見て意味のある情報が抽出できる可能性は否定できない。この研究では手術の特性類であるKervaire特性類についての情報を探索し、法コボルディズム類のKervaire障害の計算可能を通じて、偶数位数の巡回群の作用に適用するための基礎的な研究を行った。そのため、まずコンピュータによる記号計算を利用して、人手では到底不可能な計算を行い、Kervaire特性類についての予想を得て、その後に、それらの予想を証明するという手順をたどった。計算機による200次元までの計算の結果、2を法とするSteenrod代数上の関係式として2つの系列((A)および(B))が予想できた。そのうちの(A)については完全な証明がなされ、(B)についても部分的な証明は完了している。一般の関係式(B)についても完全な解決の道筋は明らかになっているので、近日中に証明が完了することが見込める。

上記の2系列の関係式のみですべてのKervaire類の関係式を尽くしているかというのは興味ある問題であるがこの問題はいわゆる「Kervaire予想」と同じかそれ以上に程度に難しいと思われる。

また、この2つの系列の関係式がAdamsスペクトル系列とどのような関連があるのかという点はホモトピー的にも意味のある問題である。Adamsスペクトル系列の高次の微分を直接決定することはほとんど不可能であるが、我々の計算の結果、分類空間 SF, F/O のホモロジー群要素の表示におけるDyer-Lashof作用素の長さがコホモロジー作用素の次元と対応しているように思われる。この点は今後も続けて研究する必要がある。

研究組織

研究代表者：　北田 泰彦　(横浜国立大学工学部教授)
研究分担者：　寺田 敏司　(横浜国立大学工学研究科教授)
研究分担者：　玉野 研一　(横浜国立大学工学部教授)
研究分担者：　平野 載倫　(横浜国立大学工学研究科教授)
研究分担者：　名倉 真紀　(横浜国立大学工学部助手)
研究分担者：　西村 尚史　(横浜国立大学教育人間科学部)

研究経費

平成 10 年度　　1,700 千円
平成 11 年度　　1,400 千円

研 究 発 表

発表論文等

1. 発表者名： T. Terada
   テーマ名： Order-preserving homeomorphisms between spaces of continuous functions
   誌名： General Topology Symposium at Kochi University
   年月日： 1998

2. 発表者名： M. Sakai, K. Tamano and Y. Yajima
   テーマ名： Regular networks for metrizable spaces and Lasnev spaces
   誌名： Bull. Polish Acad. Sci. Math.
   巻号： 46
   年月日： 1998

3. 発表者名： X. Feng and K. Tamano
   テーマ名： Countably fan-tight subspaces of a countable product of Lasnev spaces are metrizable
   誌名： Topology Proceedings
   巻号： (to appear)

4. 発表者名： K. Tamano
   テーマ名： Definitions of Sigma-spaces
   誌名： Topology Proceedings
   巻号： (to appear)

5. 発表者名： T. Nishimura
   テーマ名： Isomorphisms of smooth map germs with isomorphic local algebras
   誌名： Pitman Research Notes in Mathematical Series
   巻号： 381
   年月日： 1998

6. 発表者名： T. Terada
   テーマ名： Topological lattices $C_k(X)$ and $C_p(X)$: Embeddings and Isomorphisms
   誌名： Tsukuba J. Math
   巻号： (in press)

7. 発表者名： N. Hirano
テーマ名： Existence of entire solutions for semilinear elliptic problems on R^N
誌名： Topological Methods in Nonlinear Analysis
巻号： 13 (1)
年月日： 1999

口頭発表

1. 発表者名： 北田泰彦
テーマ名： 手術理論と Kervaire 類の関係式について
学会等名： 変換群論短期共同研究 (京都大学数理解析研究所)
年月日： 1998 年 5 月

2. 発表者名： 北田泰彦
テーマ名： Relations among smooth Kervaire classes
学会等名： 変換群論シンポジウム (山形市)
年月日： 1998 年 11 月

3. 発表者名： 北田泰彦
テーマ名： Kervaire 類の新しい関係式の予想と証明
学会等名： 変換群論短期共同研究 (京都大学数理解析研究所)
年月日： 1999 年 5 月

4. 発表者名： Y. Kitada
テーマ名： Relations among smooth Kervaire classes over the mod 2 Steenrod algebra
学会等名： First Joint Meeting Japan-Mexico in Topology and its Applications (Morelia, Mexico)
年月日： 1999 年 7 月

5. テーマ名： Stratifiable spaces, σ-spaces and μ-spaces
学会等名： First Joint Meeting Japan-Mexico in Topology and its Applications (Morelia, Mexico)
年月日： 1999 年 7 月

6. 発表者名： 寺田敏司
テーマ名： Order-preserving homeomorphisms between spaces of real-valued continuous functions.
学会等名： ジェネラルトポロジーシンポジウム (高知大学)
年月日： 1998 年 12 月

7. 発表者名： 寺田敏司
テーマ名： Embedding of $C_p(Y)$ into $C_p(X)$ as a topological sublattice
学会等名： International Conference on Topology and its Applications
年月日： 1999 年 8 月

研究の概要

1　本研究に至る経過

Milgram による分類空間 SF および F/O の結果から、ホモトピーファイブレーション

$$(CokJ)_2 \xrightarrow{i} (F/O)_2 \longrightarrow (BSO)_2$$

のファイバーについて

$$(CokJ)_2 \simeq K(\mathbf{Z}/2,6) \times_{Sq^2Sq^1} K(\mathbf{Z}/2,8) \times \cdots$$

が導かれる。すなわち、6次元の Kervaire 類 $K_6 \in H^6(F/O,\mathbf{Z}/2)$ について、$Sq^2Sq^1(i^*K_6) = 0$ が成り立つ。Madsen, May, Milgram らによる分類空間 F/O の Pontrjagin 環 $H_*(F/O,\mathbf{Z}/2)$ とその生成元の特徴付けから、F/O では関係式

$$Sq^2Sq^1K_6 + Sq^4Sq^2Sq^1K_2 = 0$$

が成り立つことが分かった。

更に、この結果を拡張して、一般に $1 < i < j$ のとき $2^i - 2$ 次元の Kervaire 類と $2^j - 2$ 次元の Kervaire 類の間に

$$Sq^{2^{i-1}}Sq^{2^{i-2}}K_{2^j-2} + Sq^{2^{j-1}}Sq^{2^{j-2}}\cdots Sq^{2^{i-1}}Sq^{2^{i-2}}K_{2^i-2} = 0$$

なる関係が成り立つことが証明できた。これを用いて、$(4k+1)$ 次元球面上の自由な involution を分類する時に、Kervaire 球面を同変に連結和することによって得られた球面は、元の球面と同変微分同相でないことが示せる。

Kervaire 類の満たす関係式を得るためには、この研究で用いた方法の他に、Kervaire 類を2次コホモロジー作用素による特性類と考えることにより、高次特性類の関係式を Adams スペクトル系列から読み取る方法がある。この方法は扱う多様体のコホモロジーの特殊性に着目する個別的な方法であり、適用可能な場合にはこの研究の成果よりもシャープな結果が得られるが、indeterminacy の影響を排除することは常に可能とは限らない。

2　本研究の方法

Kervaire 類の mod 2 Steenrod 代数を係数とした総次元 n の線形結合式

$$R = \sum Sq^{i_1} Sq^{i_2} \cdots Sq^{i_r} K_{2^j-2} \quad (i_1 + i_2 + \cdots + i_r + 2^j - 2 = n)$$

を考える。そのとき、次の 2 条件は同値である。

(1)　$R = 0$

(2)　$\forall u \in H_n(F/O, \mathbf{Z}/2), \quad \langle R,\ u \rangle = 0$

ところで、Kervaire 類は primitive であることが知れているので、(2) の u としては生成元 $u(a_1, a_2, \cdots, a_m)$, $(m \geq 2, a_{s+1} \leq a_s \leq 2a_{s+1})$ に限ってよい。ここで、$u(a_1, a_2, \cdots, a_m)$, $(m \geq 2)$ は自然な H 写像 $SF \to F/O$ による

$$Q^{a_1} Q^{a_2} \cdots Q^{a_m}[1] * [1 - 2^m] \in H_*(SF, \mathbf{Z}/2)$$

の像である。このとき、

$$\langle R, u(a_1, a_2, \cdots, a_m) \rangle = \sum \langle K_{2^j-2}, Sq_*^{i_r} \cdots Sq_*^{i_2} Sq_*^{i_1} u(a_1, a_2, \cdots, a_m) \rangle$$

となる。双対 Steenrod 作用素と Dyer-Lashof 作用素 Q^i についての西田の関係式

$$Sq_*^k Q^i = \sum_t \binom{i-k}{k-2t} Q^{i-k+t} Sq_*^t$$

を用いると、

$$Sq_*^{i_r} \cdots Sq_*^{i_2} Sq_*^{i_1} u(a_1, a_2, \cdots, a_m) = \sum u(b_1, b_2, \cdots, b_m)$$

と書けるが、Madsen らの結果から、$m \neq 2$ のとき、$\langle K_{2^j-2}, u(b_1, b_2, \cdots, b_m) \rangle = 0$ なので、始めから $m = 2$ に限ってよい。すなわち上記の (2) は

(3)　$\langle R, u(a_1, a_2) \rangle = 0, \quad (0 < a_2 \leq a_1 \leq 2a_2)$

と同値である。これを前と同様に変形すると

$$\langle R, u(a_1, a_2) \rangle = \sum \langle K_{2^j-2}, u(b_1, b_2) \rangle$$

となるが、$\langle K_{2^j-2}, u(b_1, b_2) \rangle$ が非 0 となるのは $b_2 = 2^s - 1, b_1 + b_2 = 2^j - 2$ の場合に限ることが分かっている。従って、a_1, a_2 を与えたときに $\langle R, u(a_1, a_2) \rangle$ が計算できる。

3 アルゴリズムと計算結果

1. n 次元の単項式 $R = Sq^I K_{2^j-2}$ (ただし、 $I = (i_1, i_2, \cdots, i_r)$ は admissible), と各 $u(a_1, a_2)$ ($1 < a_2 \leq a_1 \leq 2a_2, a_1 + a_2 = n$) の pairing $\langle R, u(a_1, a_2) \rangle$ を求める。これらのデータを行列と見て掃き出し法により、いくつかの R を加えたもので、常にゼロとなるものの基底を求める。この操作を低い次元から順次行う。この手順により、Kervaire 類間の関係式の基底が得られる。

2. n 次元の関係式の中で、$(n-1)$ 次元までの関係式から導かれるもの、すなわち $(n-1)$ 次元以下の関係式に Steenrod 作用素を作用させたものを除去する。残った関係式が既約な関係式であり、われわれの目標である。

ここで注意すべきことは、単項式 $R = Sq^I K_{2^j-2}$ たちにどのような順序を与えるかという点である。この順序が適切でないと、最終的に得られた結果は人の目からみて一般的な見通しがつけにくいものとなり、一般的な予想を立てることが困難となる。いくつかの試行の後に、掃き出しで残す順位は I の長さ r が短いもの、次に j が大きいものを高くしてやることで見通しのよい結果を得ることができた。以下は最終的な 200 次元までの計算の結果である。処理系としては Scheme 言語処理系として scm と hobbit scheme compiler を用いた。CPU は Celeron 466MHz, 主メモリは 256MB で、200 次元までの計算に約 3 日間を要した。

```
(A) dim=9   (Sq(4 2 1))K2 + (Sq(2 1))K6 = 0
(A) dim=16  (Sq(8 4 2))K2 + (Sq(7 3)+Sq(8 2))K6 = 0
(A) dim=17  (Sq(8 2 1))K6 + (Sq(2 1))K14 = 0
(A) dim=19  (Sq(8 4 1))K6 + (Sq(4 1))K14 = 0
(A) dim=20  (Sq(8 4 2))K6 + (Sq(4 2))K14 = 0
(B) dim=32  (Sq(16 8 2))K6 + (Sq(15 3)+Sq(16 2))K14 = 0
(A) dim=33  (Sq(16 2 1))K14 + (Sq(2 1))K30 = 0
(B) dim=34  (Sq(16 8 4))K6 + (Sq(14 6)+Sq(15 5)+Sq(16 4))K14 = 0
(A) dim=35  (Sq(16 4 1))K14 + (Sq(4 1))K30 = 0
(A) dim=36  (Sq(16 4 2))K14 + (Sq(4 2))K30 = 0
(A) dim=39  (Sq(16 8 1))K14 + (Sq(8 1))K30 = 0
(A) dim=40  (Sq(16 8 2))K14 + (Sq(8 2))K30 = 0
(A) dim=42  (Sq(16 8 4))K14 + (Sq(8 4))K30 = 0
(B) dim=64  (Sq(32 16 2))K14 + (Sq(31 3)+Sq(32 2))K30 = 0
(A) dim=65  (Sq(32 2 1))K30 + (Sq(2 1))K62 = 0
(B) dim=66  (Sq(32 16 4))K14 + (Sq(30 6)+Sq(31 5)+Sq(32 4))K30 = 0
(A) dim=67  (Sq(32 4 1))K30 + (Sq(4 1))K62 = 0
(A) dim=68  (Sq(32 4 2))K30 + (Sq(4 2))K62 = 0
(B) dim=70  (Sq(32 16 8))K14 +
             Sq(28 12)+Sq(30 10)+Sq(31 9)+Sq(32 8))K30 = 0
(A) dim=71  (Sq(32 8 1))K30 + (Sq(8 1))K62 = 0
(A) dim=72  (Sq(32 8 2))K30 + (Sq(8 2))K62 = 0
(A) dim=74  (Sq(32 8 4))K30 + (Sq(8 4))K62 = 0
(A) dim=79  (Sq(32 16 1))K30 + (Sq(16 1))K62 = 0
(A) dim=80  (Sq(32 16 2))K30 + (Sq(16 2))K62 = 0
(A) dim=82  (Sq(32 16 4))K30 + (Sq(16 4))K62 = 0
(A) dim=86  (Sq(32 16 8))K30 + (Sq(16 8))K62 = 0
(B) dim=128 (Sq(64 32 2))K30 + (Sq(63 3)+Sq(64 2))K62 = 0
(A) dim=129 (Sq(64 2 1))K62 + (Sq(2 1))K126 = 0
(B) dim=130 (Sq(64 32 4))K30 + (Sq(62 6)+Sq(63 5)+Sq(64 4))K62 = 0
(A) dim=131 (Sq(64 4 1))K62 + (Sq(4 1))K126 = 0
(A) dim=132 (Sq(64 4 2))K62 + (Sq(4 2))K126 = 0
(B) dim=134 (Sq(64 32 8))K30 +
            (Sq(60 12)+Sq(62 10)+Sq(63 9)+Sq(64 8))K62 = 0
(A) dim=135 (Sq(64 8 1))K62 + (Sq(8 1))K126 = 0
(A) dim=136 (Sq(64 8 2))K62 + (Sq(8 2))K126 = 0
(A) dim=138 (Sq(64 8 4))K62 + (Sq(8 4))K126 = 0
(B) dim=142 (Sq(64 32 16))K30 +
            (Sq(56 24)+Sq(60 20)+Sq(62 18)+Sq(63 17)+Sq(64 16))K62 = 0
(A) dim=143 (Sq(64 16 1))K62 + (Sq(16 1))K126 = 0
```

```
(A) dim=144 (Sq(64 16 2))K62 + (Sq(16 2))K126 = 0
(A) dim=146 (Sq(64 16 4))K62 + (Sq(16 4))K126 = 0
(A) dim=150 (Sq(64 16 8))K62 + (Sq(16 8))K126 = 0
(A) dim=159 (Sq(64 32 1))K62 + (Sq(32 1))K126 = 0
(A) dim=160 (Sq(64 32 2))K62 + (Sq(32 2))K126 = 0
(A) dim=162 (Sq(64 32 4))K62 + (Sq(32 4))K126 = 0
(A) dim=166 (Sq(64 32 8))K62 + (Sq(32 8))K126 = 0
(A) dim=174 (Sq(64 32 16))K62 + (Sq(32 16))K126 = 0
```

4 予想と証明

前節の計算結果から、次の2つの系列の関係式が予想される。

(A) $Sq^bSq^cK_{2^{a+1}-2}+Sq^aSq^bSq^cK_{2^a-2}=0 \qquad (a>b>c\geq 0)$

(B) $\left(Sq^{2^{a+1}}Sq^{2^b}+\sum_{i=0}^{b-1}Sq^{2^{a+1}-2^i}Sq^{2^b+2^i}\right)K_{2^{a+1}-2}+Sq^{a+1}Sq^aSq^bK_{2^a-2}=0 \quad (a>b>0)$

(A) については証明は完了しており、1999年7月モレリア市 (メキシコ) における国際会議で発表し、現在 Topology and its applications 誌に投稿中である。証明は2を法とした2項係数の場合分けを綿密に行うものである。

(B) の特別な場合: $b=a-1$ の場合については証明は (A) の証明より簡単であり、既に完了している。(B) の一般の場合についても (A) の場合に準備した補題群にあと一つの補題を準備しており、後は (A) と同様に証明できるものと思われる。

(A) 系列関係式の証明

RELATIONS OF SMOOTH KERVAIRE CLASSES OVER THE MOD 2 STEENROD ALGEBRA

YASUHIKO KITADA

In memory of Professor Katsuo Kawakubo

ABSTRACT. In the surgery theory of smooth smooth manifolds, it is often difficult to determine the existence or the non-existence of a smooth normal map with nontrivial surgery obstructions. We present and prove a simple relation over the mod 2 Steenrod algebra between two smooth Kervaire classes in different dimensions. This formula enables us to compute the Kervaire surgery invariants for various manifolds.

1. INTRODUCTION AND RESULT

Let M^n be a smooth closed manifold. Then a normal cobordism class of a normal map with target M can be represented by a map

$$f : M^n \longrightarrow F/O$$

where

$$F/O \longrightarrow BSO \longrightarrow BSF$$

is a fibration of infinite loop spaces. Here BSO is the classifying space of the oriented stable orthogonal vector bundles and BSF is the classifying space of stable oriented spherical fibrations.

Under these circumstances, the Kervaire obstruction of the surgery datum f is given by

$$c(f) = \langle V(M)^2 \sum_{i \geq 2} f^* K_{2^i-2}, [M]_2 \rangle,$$

where $V(M)$ is the total Wu class of M^n, K_{2^i-2} is the smooth Kervaire class which lies in $H^{2^i-2}(F/O; \mathrm{Z}/2)$, and $[M]_2$ is the mod 2 homology fundamental class of M.

When we study the Kervaire obstruction map for a given manifold M^n, it is important to know whether the the Kervaire obstruction map

$$c \ : \ [M^n, F/O] \longrightarrow \mathrm{Z}/2$$

is trivial or not. The most difficult case is when M is a sphere; the so-called Kervaire invariant conjecture is still an open question in dimensions $2^k - 2$. In some cases of manifolds, the answers are known. When M^n is the real projective spaces of dimension $4k+2$, the Kervaire obstruction map is always

1991 *Mathematics Subject Classification.* 55R55,55R67,57S10,57S12.
Key words and phrases. surgery, Kervaire class.

nontrivial([1]). When M^n $(n = 4k+2)$ is a product of an odd-dimensional sphere $S^{2p+1}$and an odd-dimensional real projective space RP^{2q+1} $(p \leq q)$, the Kervaire surgery obstruction map is trivial unless $n+2$ is a power of 2. This result was obtained by making use of a relation between two Kervaire classes:

$$Sq^{2^{r-1}}Sq^{2^{r-2}}\cdots Sq^{2^{s-1}}Sq^{2^{s-2}}K_{2^s-2} = Sq^{2^{s-1}}Sq^{2^{s-2}}K_{2^r-2}, \quad (r > s \geq 2).$$

As a geometric application of this result, we were able to prove that the connected sum of a real projective space RP^{4k+1} with a Kervaire homotopy sphere that bounds a framed manifold with Kervaire invariant one is not diffeomorphic to the original projective space ([2]). This relation, however, can be deduced from the following stronger but simpler new formula:

Theorem 1.1. *For the smooth Kervaire classes we have a relation*

$$Sq^{2^r}Sq^{2^s}Sq^{2^t}K_{2^r-2} = Sq^{2^s}Sq^{2^t}K_{2^{r+1}-2}.$$

where r, s and t are integers that satisfy $r > s > t \geq 0$.

2. Preliminaries

Before we go into the details of proofs, we present the machineries which will be used in the proofs. In this paper all homologies and cohomologies are in coefficients mod 2 and will be omitted from the notation.

The mod 2 homology and cohomology of the classifying space of surgery F/O were studied by Milgram [5] and others. The notation we adopt in this paper is contained in [4]. The mod 2 Steenrod squaring operations on the cohomology group $H^*(F/O)$ is not directly calculable. However, when we consider its dual $H_*(F/O)$, using the infinite loop space structure of the classifying spaces, we know its algebra generators as follows.

Let $I = (i_1, i_2, \cdots, i_n)$ be a finite sequence of non-negative integers. We shall write Q^I to be the composite of mod 2 Dyer-Lashof homology operations $Q^{i_1}Q^{i_2}\cdots Q^{i_n}$. We say that I or Q^I is allowable if $i_j \leq 2i_{j+1}$ holds for all j, $1 \leq j \leq n-1$. Define its length $l(I) = n$ and its excess $e(I)$ by

$$e(I) = \sum_{j=1}^{n-1}(i_j - 2i_{j+1}) + i_n = i_1 - i_2 - \cdots - i_n.$$

The Pontrjagin ring of SF is known as follows.

Theorem 2.1 (Madsen-Milgram)**.**

$$H_*(SF) = E\{Q^i[1] * [-1] | i \geq 1\} \otimes P\{Q^iQ^i[1] * [-3] | i \geq 1\}$$
$$\otimes P\{Q^I[1] * [1-2^n] | I : \text{allowable}, l(I) = n \geq 2,\ e(I) \geq 1, i_n \geq 1\}$$

Here the Dyer-Lashof operation Q^i s is based on infinite loop structure of $\Omega^\infty S^\infty$ and not the H-space structure of SF induced by composition of maps, and $H_*(SF)$ is considered as a subalgebra of $H_*(\Omega^\infty S^\infty)$. There is another choice of generators; one can take elements of the form

$$\hat{Q}^{i_1}\hat{Q}^{i_2}\ldots\hat{Q}^{i_{n-2}}(Q^{i_{n-1}}Q^{i_n}[1] * [-3])$$

as alternative generators as well, where $\hat{Q}^i$ is based on the composition product on SF. However, this choice of generators does not affect the arguments that follow.

The natural map $SF \longrightarrow F/O$ in the sequence of fibrations

$$SO \longrightarrow SF \longrightarrow F/O \longrightarrow BSO \longrightarrow BSF,$$

allows us to identify $H_*(F/O)$ with the subalgebra of $H_*(SF)$:

$$H_*(F/O) = P\{Q^iQ^i[1] * [-3] | i \geq 1\}$$
$$\otimes P\{Q^I[1] * [1-2^n] | I : \text{allowable}, l(I) = n \geq 2,\ e(I) \geq 1, i_n \geq 1\}.$$

As to the homology operation, we have Adem relation

$$Q^aQ^b = \sum_i \binom{i-b-1}{2i-a} Q^{a+b-i}Q^i \quad \text{for } a > 2b$$

and Nishida relation [6]

$$Sq^a_* Q^b = \sum_i \binom{b-a}{a-2i} Q^{b-a+i} Sq^i_*,$$

where Sq^i_* denotes the dual of the Steenrod squaring operation Sq^i.

In what follows, given a sequence $I = (i_1, i_2, \cdots, i_n)$ (not necessarily allowable), we shall also write $u(i_1, i_2, \cdots, i_n)$ to represent the element $Q^I[1] * [1-2^n]$ either in $H_*(SF)$ or $H_*(F/O)$.

As to the characterization of the cohomology smooth Kervaire classes, we have the following theorem.

Theorem 2.2. *For $I = (i_1, i_2, \cdots, i_n)$, $\langle K_{2^{q+1}-2}, u(i_1, i_2, \cdots, i_n)\rangle$ is nonzero if and only if $n = 2$, $i_1 + i_2 = 2^{q+1} - 2$ and $i_2 = 2^k - 1$ for some k, $0 < k \leq q$.*

All the binomial coefficients in this paper are considered mod 2. In the proof of our main theorem, we shall encounter many binomial coefficients. The following is the most fundamental criterion for determining the modulo 2 value of the binomial coeffcient ([7], I.2.6.LEMMA).

Lemma 2.1. *Let*

$$a = \sum_{i \geq 0} a_i\ 2^i, \quad b = \sum_{i \geq 0} b_i\ 2^i$$

be 2-adic expansions of non-negative integers a and b, where a_i and b_i's are 0 or 1. Then the binomial coefficient $\binom{a}{b}$ is 0 if and only if there exists an i such that $a_i = 0$ and $b_i = 1$.

For a, b not satisfying $a \geq b \geq 0$, we use the convention $\binom{a}{b} = 0$.

From Lemma 2.1 we have two lemmas below:

Lemma 2.2. *The mod 2 binomial coefficient $\binom{a}{b}$ $(a \geq b \geq 0)$ is characterized by the following properties:*

1) $\binom{a}{b} = 1$ *if* $b = 0$.
2) $\binom{a}{b} = 0$ *if a is even and b is odd.*

3) *else* $\binom{a}{b} = \binom{a'}{b'}$ *where* $a' = [a/2]$ *and* $b' = [b/2]$, *where* $[x]$ *denotes the largest integer not exceeding* x.

Lemma 2.3. *Let* $a = c * 2^n + d$ *and* $b < 2^n$, *where* c *is a positive integer and* d *satisfies* $0 \leq d < 2^n$. *Then we have* $\binom{a}{b} = \binom{d}{b}$.

The following lemmas will be frequently useful in the proofs. All the proofs are performed by inductions.

Lemma 2.4. *Let* $a \geq 0$. $\binom{2^a+i}{2i+1}$ *is nonzero if and only if* $i = 2^a - 1$.

Proof. We use induction on a. The case $a = 0$ is easy. Let $a > 0$ and assume the conclusion is true for $a-1$. We assume that $\binom{2^a+i}{2i+1} \neq 0$. If i is even, then we have $\binom{2^a+i}{2i+1} = 0$ by Lemma 2.1. Hence i should be odd. Put $i = 2j + 1$. Then we have $\binom{2^a+i}{2i+1} = \binom{2^a+2j+1}{4j+3} = \binom{2^{a-1}+j}{2j+1}$. By inductive assumption, we have $j = 2^{a-1} - 1$. Therefore we have $i = 2^a - 1$. □

Lemma 2.5. *Let* $a \geq 0$. $\binom{2^a+i}{2i}$ *is nonzero if and only if* $i = 2^a - 2^k$ $(0 \leq k \leq a)$ *or* $i = 2^a$.

Proof. The result is true for $a = 0$. Assume that $a > 0$ and the conclusion holds for $a - 1$.
Case $i = 2j$: $\binom{2^a+i}{2i} = \binom{2^a+2j}{4j} = \binom{2^{a-1}+j}{2j}$. This is nonzero if and only if $j = 2^{a-1} - 2^l$ $(0 \leq l \leq a-1)$ or $j = 2^{a-1}$. That is $i = 2^a - 2^k$ $(1 \leq k \leq a)$ or $i = 2^a$.
Case $i = 2j + 1$: $\binom{2^a+i}{2i} = \binom{2^a+2j+1}{4j+2} = \binom{2^{a-1}+j}{2j+1}$. By the previous lemma, we have $j = 2^{a-1} - 1$. Therefore $i = 2^a - 1 = 2^a - 2^0$. □

Lemma 2.6. *Let* $a \geq 0$. $\binom{2^a-1+i}{2i}$ *is nonzero if and only if* $i = 2^a - 2^k$ $(0 \leq k \leq a)$.

Proof. Induction on a : For $a = 0$, the assetion is clear. Suppose that the assertion holds below a with $a \geq 1$.
Case $i = 2j$: We have $\binom{2^a-1+2j}{4j} = \binom{2^{a-1}-1+j}{2j}$ which is nonzero if and only if $j = 2^{a-1} - 2^l$ for some l, $0 \leq l \leq a-1$ by the inductive assumption. That is $i = 2^a - 2^k$ $(1 \leq k \leq a)$.
Case $i = 2j + 1$: We have $\binom{2^a-1+(2j+1)}{4j+2} = \binom{2^{a-1}+j}{2j+1}$ which is nonzero if and only if $j = 2^{a-1} - 1$ by Lemma 2.4. That is $i = 2^a - 1$. □

Lemma 2.7. *Let* $a \geq 0$. $\binom{2^a-1+i}{2i+1}$ *is nonzero if and only if* $i = 2^a - 2^k$ $(1 \leq k \leq a)$.

Proof. Induction on a : The case $a = 0$ is clear. Let $a \geq 1$, then $\binom{2^a-1+i}{2i+1}$ is zero for i odd. Put $i = 2j$, then we have $\binom{2^a-1+i}{2i+1} = \binom{2^a-1+2j}{4j+1} = \binom{2^{a-1}-1+j}{2j}$, which is nonzero if and only if $j = 2^{a-1} - 2^l$ with $0 \leq l \leq a-1$, by Lemma 2.6. That is $i = 2^a - 2^k$ $(1 \leq k \leq a)$. □

Lemma 2.8. *Let* $a > b \geq 0$. $\binom{2^a+2^b+i}{2i+1}$ *is nonzero if and only if* $i = (2^a - 2^k) + 2^b - 1$ $(b+1 \leq k \leq a)$ *or* $i = 2^a + 2^b - 1$.

Proof. Induction on b : When $b = 0$, i should be even. So we may put $i = 2j$. We have $\binom{2^a+1+2j}{4j+1} = \binom{2^{a-1}+j}{2j}$ which is nonzero if and only if $j = 2^{a-1} - 2^l$ $(0 \leq l \leq a-1)$ or $j = 2^{a-1}$ by Lemma 2.5. That is $i = 2^a - 2^k$ $(1 \leq k \leq a)$ or $i = 2^a$. So the result holds for $b = 0$. Let $b \geq 1$. If i is even, $\binom{2^a+2^b+i}{2i+1}$ is zero. So we may put $i = 2j + 1$. We have $\binom{2^a+2^b+2j+1}{4j+3} = \binom{2^{a-1}+2^{b-1}+j}{2j+1}$, which is nonzero if and only if $j = (2^{a-1} - 2^l) + 2^{b-1} - 1$ or $j = 2^{a-1} + 2^{b-1} - 1$ by the inductive assumption. That is $i = 2^a - 2^k + 2^b - 1$ $(b + 1 \leq k \leq a)$ or $i = 2^a + 2^b - 1$. □

Lemma 2.9. *Let $a > b \geq 0$. $\binom{2^a-2^b+i}{2i+1}$ is nonzero if and only if $i = (2^a - 2^k) + 2^b - 1$ $(b + 1 \leq k \leq a)$.*

Proof. Induction on b : The case when $b = 0$ is Lemma 2.7. Let $b \geq 1$ and since $\binom{2^a-2^b+i}{2i+1}$ is zero for i even, we put $i = 2j + 1$. We have $\binom{2^a-2^b+2j+1}{4j+3} = \binom{2^{a-1}-2^{b-1}+j}{2j+1}$, which is nonzero if and only if $j = (2^{a-1} - 2^l) + 2^{b-1} - 1$ $(b \leq l \leq a - 1)$ by the inductive assumption. That is $i = (2^a - 2^k) + 2^b - 1$ $(b + 1 \leq k \leq a)$. □

Lemma 2.10. *Let $a > b \geq 0$. $\binom{2^a-2^b+i}{2i}$ is nonzero if and only if $i = (2^a - 2^k) + (2^b - 2^l)$ $(b + 1 \leq k \leq a, 0 \leq l \leq b)$ or $i = 2^a - 2^b$.*

Proof. When $b = 0$, the result follows from Lemma 2.6. Let $b \geq 1$.
Case $i = 2j$: We have $\binom{2^a-2^b+2j}{4j} = \binom{2^{a-1}-2^{b-1}+j}{2j}$ which is nonzero if and only if $j = (2^{a-1} - 2^p) + (2^{b-1} - 2^q)$ where $b \leq p \leq a-1$, $0 \leq q \leq b-1$ by the inductive assumption. That is $i = (2^a - 2^k) + (2^b - 2^l)$ where $b + 1 \leq k \leq a$, $1 \leq l \leq b - 1$.
Case $i = 2j + 1$: We have $\binom{2^a-2^b+2j+1}{4j+2} = \binom{2^{a-1}-2^{b-1}+j}{2j+1}$, which is nonzero if and only if $j = (2^{a-1} - 2^p) + 2^{b-1} - 1$ $(b \leq p \leq a - 1)$ by Lemma 2.9. That is $i = 2^a - 2^k + 2^b - 1$, $(b + 1 \leq k \leq a)$. □

Lemma 2.11. *Let $a > b \geq 0$. $\binom{2^a+2^b-1+i}{2i}$ is nonzero if and only if $i = (2^a - 2^k) + (2^b - 2^l)$, $i = 2^a + (2^b - 2^l)$ or $i = 2^a - 2^b$ where $b + 1 \leq k \leq a$, $0 \leq l \leq b$.*

Proof. Induction on b : When $b = 0$, $\binom{2^a+i}{2i}$ is nonzero if and only if $i = 2^a - 2^k$ $(0 \leq k \leq a)$ or $i = 2^a$ by Lemma 2.5. Let $b \geq 1$.
Case $i = 2j$: We have $\binom{2^a+2^b-1+2j}{4j} = \binom{2^{a-1}+2^{b-1}-1+j}{2j}$, which is nonzero if and only if $j = 2^{a-1} - 2^{b-1}$, $j = 2^{a-1} - 2^p + 2^{b-1} - 2^q$ or $j = 2^{a-1} + (2^{b-1} - 2^q)$ where $b \leq p \leq a - 1$, $0 \leq q \leq b - 1$ by the inductive assumption. That is $i = 2^a - 2^b$, $i = 2^a - 2^k + 2^b - 2^l$ or $i = 2^a + 2^b - 2^l$ where $b + 1 \leq k \leq a$, $1 \leq l \leq b$.
Case $i = 2j+1$: We have $\binom{2^a+2^b+2j}{4j+2} = \binom{2^{a-1}+2^{b-1}+j}{2j+1}$, which is nonzero if and only if $j = 2^{a-1} + 2^{b-1} - 1$ or $j = (2^{a-1} - 2^p) + 2^{b-1} - 1$ where $b \leq p \leq a - 1$ by Lemma 2.8. That is $i = 2^a + 2^b - 1$ or $i = (2^a - 2^k) + 2^b - 1$ where $b + 1 \leq k \leq a$. □

Lemma 2.12. *Let $a > b \geq 0$. $\binom{2^a+2^b-1+i}{2i+1}$ is nonzero if and only if $i = 2^a - 2^b$, $i = (2^a - 2^k) + (2^b - 2^l)$ or $i = 2^a + (2^b - 2^l)$ where $b+1 \leq k \leq a$, $1 \leq l \leq b$.*

Proof. Induction on b : The assertion holds for $b = 0$ by Lemma 2.4. We let $b \geq 1$, then we have only to consider the case when i is even. Putting $i = 2j$, we have $\binom{2^a+2^b-1+2j}{4j+1} = \binom{2^{a-1}+2^{b-1}-1+j}{4j}$, which is nonzero if and only if $j = 2^{a-1} - 2^{b-1}$, $j = 2^{a-1} - 2^p + 2^{b-1} - 2^q$ or $j = 2^{a-1} + 2^{b-1} - 2^q$ where $b \leq p \leq a-1$ and $0 \leq q \leq b-1$ by the inductive assumption. That is $i = 2^a - 2^b$, $i = 2^a - 2^k + 2^b - 2^l$ or $i = 2^a + 2^b - 2^l$ where $b+1 \leq k \leq a$ and $1 \leq l \leq b$. □

Lemma 2.13. *Let $a > b \geq 0$. $\binom{2^a-2^b-1+i}{2i}$ is nonzero if and only if $i = (2^a - 2^k) + (2^b - 2^l)$ or $i = (2^a - 2^m) + 2^b$ for $b+1 \leq k \leq a$, $0 \leq l \leq b$, $b+2 \leq m \leq a$.*

Proof. Induction on b : If $b = 0$ the result follows from Lemma 2.10. Let $b \geq 1$.
Case $i = 2j$: We have $\binom{2^a-2^b-1+2j}{4j} = \binom{2^{a-1}-2^{b-1}-1+j}{2j}$, which is nonzero if and only if $i = 2^{a-1} - 2^p + 2^{b-1} - 2^q$ $(b \leq p \leq a-1, 0 \leq q \leq b-1)$ or $i = 2^{a-1} - 2^\mu + 2^{b-1}$ $(b+1 \leq \mu \leq a-1)$. That is $i = 2^a - 2^k + 2^b - 2^l$ $(b+1 \leq k \leq a, 1 \leq l \leq b)$ or $2^a - 2^m + 2^b$ $(b+2 \leq m \leq a)$.
Case $i = 2j+1$: We have $\binom{2^a-2^b+2j}{4j+1} = \binom{2^{a-1}-2^{b-1}+j}{2j}$, which is nonzero, by Lemma 2.9, if and only if $j = 2^{a-1} - 2^p + 2^{b-1} - 1$. That is $i = 2^a - 2^k + 2^b - 1$ where $b+1 \leq k \leq a$. □

Lemma 2.14. *Let $a > b > c \geq 0$. $\binom{2^a+2^b-2^c+i}{2i+1}$ is nonzero if and only if $i = 2^a - 2^b + 2^c - 1$, $i = (2^a - 2^k) + (2^b - 2^l) + 2^c - 1$ or $i = 2^a + (2^b - 2^l) + 2^c - 1$ where $b+1 \leq k \leq a$ and $c+1 \leq l \leq b$.*

Proof. Induction on c : The result for $c = 0$ is immediate from Lemma 2.12. So we let $c \geq 1$. We have only to consider the case when i is odd and put $i = 2j+1$. Then we have $\binom{2^a+2^b-2^c+2j+1}{4j+3} = \binom{2^{a-1}+2^{b-1}-2^{c-1}+j}{2j+1}$, which is nonzero if and only if $j = 2^{a-1} - 2^{b-1} + 2^{c-1} - 1$, $j = 2^{a-1} - 2^p + 2^{b-1} - 2^q + 2^{c-1} - 1$ or $j = 2^{a-1} + 2^{b-1} - 2^l + 2^{c-1} - 1$ where $b \leq p \leq a-1$ and $c \leq q \leq b-1$ by the inductive assumption. That is $i = 2^a - 2^b + 2^c - 1$, $i = 2^a - 2^k + 2^b - 2^l + 2^c - 1$ or $i = 2^a + 2^b - 2^l + 2^c - 1$ where $b+1 \leq k \leq a$ and $c+1 \leq l \leq b$. □

Lemma 2.15. *Let $a > b > c \geq 0$. $\binom{2^a+2^b-2^c+i}{2i}$ is nonzero if and only if $i = (2^a - 2^k) + (2^b - 2^l) + (2^c - 2^m)$, $i = 2^a + (2^b - 2^l) + (2^c - 2^m)$, $i = 2^a - 2^b + (2^c - 2^m)$, $i = (2^a - 2^k) + 2^b - 2^c$ or $i = 2^a + 2^b - 2^c$ where $b+1 \leq k \leq a$, $c+1 \leq l \leq b$, $0 \leq m \leq c$.*

Proof. Induction on c : The result for $c = 0$ follows from Lemma 2.11. We consider the case $c \geq 1$.

Case $i = 2j$: We have $\binom{2^a+2^b-2^c+2j}{4j} = \binom{2^{a-1}+2^{b-1}-2^{c-1}+j}{2j}$. By the inductive assumption this is nonzero if and only if

$$j = \begin{cases} 2^{a-1} - 2^p + 2^{b-1} - 2^q + 2^{c-1} - 2^\mu \\ 2^{a-1} + 2^{b-1} - 2^q + 2^{c-1} - 2^\mu \\ 2^{a-1} - 2^{b-1} + 2^{c-1} - 2^\mu \\ 2^{a-1} + 2^{b-1} - 2^{c-1} \\ 2^{a-1} - 2^p + 2^{b-1} - 2^{c-1} \end{cases}$$

where $b \le p \le a-1$, $c \le q \le b-1$ and $0 \le \mu \le c-1$. That is

$$i = \begin{cases} 2^a - 2^k + 2^b - 2^l + 2^c - 2^m \\ 2^a + 2^b - 2^l + 2^c - 2^m \\ 2^a - 2^b + 2^c - 2^m \\ 2^a + 2^b - 2^c \\ 2^a - 2^k + 2^b - 2^c \end{cases}$$

where $b+1 \le k \le a$, $c+1 \le l \le b$ and $1 \le m \le c$.
Case $i = 2j+1$: We have $\binom{2^a+2^b-2^c+2j+1}{4j+2} = \binom{2^{a-1}+2^{b-1}-2^{c-1}+j}{2j+1}$. By Lemma 2.14, this is nonzero if and only if

$$j = \begin{cases} 2^{a-1} - 2^{b-1} + 2^{c-1} - 1 \\ 2^{a-1} - 2^p + 2^{b-1} - 2^q + 2^{c-1} - 1 \\ 2^{a-1} + 2^{b-1} - 2^q + 2^{c-1} - 1 \end{cases}$$

where $b \le p \le a-1$ and $c \le q \le b-1$. That is $i = 2^a - 2^b + 2^c - 1$, $i = 2^a - 2^k + 2^b - 2^l + 2^c - 1$ or $i = 2^a + 2^b - 2^l + 2^c - 1$. □

Lemma 2.16. *Let $a > b > c \ge 0$. $\binom{2^a+2^b-2^c-1+i}{2i}$ is nonzero if and only if $i = (2^a - 2^k) + (2^b - 2^l) + (2^c - 2^m)$, $i = 2^a + (2^b - 2^l) + (2^c - 2^m)$, $i = 2^a - 2^b + (2^c - 2^m)$, $i = 2^a - 2^b + 2^c$, $i = (2^a - 2^k) + (2^b - 2^q) + 2^c$ or $i = 2^a + (2^b - 2^q) + 2^c$ where $b+1 \le k \le a$, $c+1 \le l \le b$, $0 \le m \le c$ and $c+2 \le q \le b$.*

Proof. Induction on c : The result for $c = 0$ follows from Lemma 2.15and we consider the case $c \ge 1$.
$i = 2j$: We have $\binom{2^a+2^b-2^c-1+2j}{4j} = \binom{2^{a-1}+2^{b-1}-2^{c-1}-1+j}{2j}$ which is, by the inductive assumption, nonzero if and only if

$$j = \begin{cases} 2^{a-1} - 2^\alpha + 2^{b-1} - 2^\beta + 2^{c-1} - 2^\delta \\ 2^{a-1} + 2^{b-1} - 2^\beta + 2^{c-1} - 2^\delta \\ 2^{a-1} - 2^{b-1} + 2^{c-1} - 2^\delta \\ 2^{a-1} - 2^{b-1} + 2^{c-1} \\ 2^{a-1} - 2^\alpha + 2^{b-1} - 2^\gamma + 2^{c-1} \\ 2^{a-1} + 2^{b-1} - 2^\gamma + 2^{c-1} \end{cases}$$

where $b \leq \alpha \leq a-1$, $c \leq \beta \leq b-1$, $0 \leq \delta \leq c-1$ and $c+1 \leq \gamma \leq b-1$. That is

$$i = \begin{cases} 2^a - 2^k + 2^b - 2^l + 2^c - 2^m \\ 2^a + 2^b - 2^l + 2^c - 2^m \\ 2^a - 2^b + 2^c - 2^m \\ 2^a - 2^b + 2^c \\ 2^a - 2^k + 2^b - 2^q + 2^c \\ 2^a + 2^b - 2^q + 2^c \end{cases}$$

where $b+1 \leq k \leq a$, $c+1 \leq l \leq b$, $1 \leq m \leq c$ and $c+2 \leq q \leq b$.
Case $i = 2j+1$: We have $\binom{2^a+2^b-2^c+2j}{4j+2} = \binom{2^{a-1}+2^{b-1}-2^{c-1}+j}{2j+1}$. By Lemma 2.14, this is nonzero if and only if $j = 2^{a-1} - 2^{b-1} + 2^{c-1} - 1$, $j = 2^{a-1} - 2^\alpha + 2^{b-1} - 2^\beta + 2^{c-1} - 1$, $2^{a-1} + 2^{b-1} - 2^\beta + 2^{c-1} - 1$ where $b \leq \alpha \leq a-1$ and $c \leq \beta \leq b-1$. That is $i = 2^a - 2^b + 2^c - 1$, $i = 2^a - 2^k + 2^b - 2^l + 2^c - 1$ and $i = 2^a + 2^b - 2^l + 2^c - 1$ where $b+1 \leq k \leq a$ and $c+1 \leq l \leq b$. □

3. Proofs of Theorems

We begin with the proof of Theorem 2.2.

Let $I = (i_1, i_2, \cdots, i_n)$, $(i_n > 0)$ and consider $u_I = u(i_1, i_2, \cdots, i_n)$. When I is allowable, then we already know that the pairing $\langle K_{2^{q+1}-2}, u_I \rangle$ is nonzero if and only if $n = 2$ and $i_1 = i_2 = 2^q - 1$, ([3],[2]). Suppose that I is not allowable. Then we can use the Adem relation and express u_I as a sum of allowable terms u_J. Since the Adem relation preserves the length of the indices, $\langle K_{2^{q+1}-2}, u_I \rangle$ is zero if $l(I) \neq 2$. Let $I = (i_1, i_2)$ be non-allowable. In this case, by the Adem relation we have

$$u(i_1, i_2) = \sum_j \binom{j - i_2 - 1}{2j - i_1} u(2^{q+1} - 2 - j, j).$$

The pairing $\langle K_{2^{q+1}-2}, u(i_1, i_2) \rangle$ is nonzero if and only if the binomial coefficient $\binom{j-i_2-1}{2j-i_1}$ is nonzero for $j = 2^q - 1$. Then we have $\binom{j-i_2-1}{2j-i_1} = \binom{2^q-i_2-2}{i_2} = \binom{2^q-i_2-2}{2^q-2i_2-2} = \binom{2^{q-1}-1+(2^{q-1}-i_2-1)}{2(2^{q-1}-i_2-1)}$. By Lemma 2.6, this is nonzero if and only if $2^{q-1} - i_2 - 1 = 2^{q-1} - 2^k$ for some $k \leq q-1$, i.e. $i_2 = 2^k - 1$. This completes the proof.

The term of the form $u(2^{q+1} - 2^k - 1, 2^k - 1)$ that appeared in the above proof will be called a Kervaire dual in this paper.

The proof of Theorem 1.1 is much more complicated. We shall make a few observations. The statement

$$Sq^{2^r} Sq^{2^s} Sq^{2^t} K_{2^r-2} = Sq^{2^s} Sq^{2^t} K_{2^{r+1}-2}$$

is equivalent to the statement that

$$\langle Sq^{2^r} Sq^{2^s} Sq^{2^t} K_{2^r-2}, u \rangle = \langle Sq^{2^s} Sq^{2^t} K_{2^{r+1}-2}, u \rangle$$

holds for any $u \in H_{2^{r+1}+2^s+2^t-2}(F/O)$.

We may assume that the homology class u is non-decomposable since the Kervaire classes are primitive and the Steenrod squaring operation maps primitives to primitives. Thus the above statement is equivalent to

$$\langle K_{2^r-2}, Sq_*^{2^t} Sq_*^{2^s} Sq_*^{2^r} u_I \rangle = \langle K_{2^{r+1}-2}, Sq_*^{2^t} Sq_*^{2^s} u_I \rangle,$$

where Sq_*^i's are the dual operations on homology and u_I is the generator of the polynomial algebra $H_*(F/O)$. By the Nishida relation, we see that Sq_*^i preserves the length of u_I and therefore we may assume that the length of I is two.

Let $I = (a, b)$ be allowable i.e. $0 < b \leq a \leq 2b$ and $a + b = 2^{r+1} + 2^s + 2^t - 2$. We have to count the occurrences of Kervaire duals of the form $u(2^r - 2^m - 1, 2^m - 1)$ in $Sq_*^{2^t} Sq_*^{2^s} Sq_*^{2^r} u(a, b)$ and $u(2^{r+1} - 2^k - 1, 2^k - 1)$ in $Sq_*^{2^t} Sq_*^{2^s} u(a, b)$ and show that the occurrences in both cases coincide for all $u(a, b)$.

Let $r > s > t \geq 0$ and $a+b = 2^{r+1}+2^s+2^t-2$ with allowability condition $0 < b \leq a \leq 2b$.

First, repeated use of the Nishida relation shows that

$$Sq_*^{2^t} Sq_*^{2^s} Sq_*^{2^r} u(a, b) = \sum_{i,j,k} ABCDEFu(c, d)$$

where

$$\begin{aligned}
A &= \binom{a - 2^r}{2^r - 2i} = \binom{2^r + 2^s + 2^t - 2 - b}{2^r - 2i}, \\
B &= \binom{b - i}{i}, \\
C &= \binom{a - 2^r - 2^s + i}{2^s - 2j} = \binom{2^r + 2^t - 2 - (b - i)}{2^s - 2j}, \\
D &= \binom{b - i - j}{j}, \\
E &= \binom{a - 2^r - 2^s - 2^t + i + j}{2^t - 2k} = \binom{2^r - 2 - (b - i - j)}{2^t - 2k}, \\
F &= \binom{b - i - j - k}{k}, \\
c &= a - 2^r - 2^s - 2^t + i + j + k, \quad d = b - i - j - k.
\end{aligned}$$

We count the occurrences of $u(c, d)$ with $d = 2^m - 1$, $(m \leq r - 1)$. From $3b \geq a+b = 2^{r+1}+2^s+2^t-2$, we have $3(2^m-1+i+j+k) \geq 2^{r+1}+2^s+2^t-2$, that is,

$$3(i + j + k) \geq 2^{r+1} + 2^s + 2^t + 1 - (2^{m+1} + 2^m). \tag{3.1}$$

If $F \neq 0$, then we have $0 \leq k \leq 2^m - 1$. If $A \neq 0$, then we have $i \leq 2^{r-1}$. If $C \neq 0$, then we have $j \leq 2^{s-1}$. Combining these, we have from (3.1),

$$3(2^{r-1} + 2^{s-1} + 2^m - 1) \geq 2^{r+1} + 2^s + 2^t + 1 - (2^{m+1} + 2^m).$$

Therefore we get

$$2^{m+2} + 2^{m+1} + 2^{s-1} \geq 2^{r-1} + 2^t + 4.$$

This inequality is impossible if $m \leq r - 5$. Thus we may limit the range of m as $r - 4 \leq m \leq r - 1$.

Lemma 3.1. *If $r \geq t + 3$, there are no occurrences of Kervaire duals with $m = r - 4$.*

Proof. Let $r \geq t + 3$ and $m = r - 4$. From (3.1) we have

$$3(i+j+k) \geq 2^{r+1}+2^s+2^t+1-(2^{r-3}+2^{r-4}) = 2^r+2^{r-1}+2^{r-2}+2^{r-4}+2^s+2^t+1.$$

On the other hand, we have

$$3(i + j + k) \leq 3(2^{r-1} + 2^{s-1} + 2^{t-1}) = 2^r + 2^{r-1} + 2^s + 2^{s-1} + 2^t + 2^{t-1}.$$

From these two inequalities we have

$$2^r + 2^{r-1} + 2^s + 2^{s-1} + 2^t + 2^{t-1} \geq 2^r + 2^{r-1} + 2^{r-2} + 2^{r-4} + 2^s + 2^t + 1,$$

that is

$$2^{s-1} + 2^{t-1} \geq 2^{r-2} + 2^{r-4} + 1.$$

However this is impossible since

$$2^{s-1} + 2^{t-1} \leq 2^{r-2} + 2^{r-4}.$$

□

We shall count the occurrences of Kervaire duals in three different cases. First we begin with the case when $r \geq s + 2$.

Lemma 3.2. *Let $r \geq s+2$. Then a Kervaire dual appears for $m = r - 1$ if and only if*

$$(a, b) = \begin{cases} (2^r + 2^s - 1, 2^r + 2^t - 1) \\ (2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1, 2^r - 2^t + 2^x + 2^y - 1) \\ \qquad (t + 1 \leq x \leq s - 1,\ 0 \leq y \leq t). \end{cases}$$

Proof. We look for the condition for $ABCDEF \neq 0$. $F \neq 0$ if and only if $0 \leq k \leq 2^{r-1} - 1$. $E = \binom{2^{r-1}-1-k}{2^t-2k}$ is nonzero for $k = 0$.
Case $t = 0$: E is nonzero if and only if $k = 0$.
Case $t \geq 1$: We see by Lemma 2.13 that

$$E = \binom{2^{r-1} - 2^{t-1} - 1 + (2^{t-1} - k)}{2(2^{t-1} - k)}$$

is nonzero if and only if

$$2^{t-1} - k = \begin{cases} (2^{r-1} - 2^\mu) + 2^{t-1} & (t + 1 \leq \mu \leq r - 1) \\ (2^{r-1} - 2^\kappa) + (2^{t-1} - 2^l) & (t \leq \kappa \leq r - 1,\ 0 \leq l \leq r - 1). \end{cases}$$

Hence $k = 0$ or $k = 2^l$ $(0 \le l \le t-1)$. Next we consider the condition for $D = \binom{2^{r-1}+k-1}{j}$ to be nonzero. When $k = 0$, $D \neq 0$ if and only if $0 \le j \le 2^{r-1} - 1$. When $k = 2^l$ $(0 \le l \le t-1)$, $D \neq 0$ if and only if $0 \le j \le 2^l - 1$ since we may assume that $j \le 2^{s-1}$ in view of C. To find the condition for $C \neq 0$, we must consider several cases.
Case $k = 0$ and $0 \le j \le 2^{r-1} - 1$:

$$\begin{aligned} C &= \binom{2^{r-1} + 2^t - 1 - j}{2^s - 2j} \\ &= \binom{2^{r-1} - 2^{s-1} + 2^t - 1 + (2^{s-1} - j)}{2(2^{s-1} - j)} \\ &= \binom{2^{r-2} + 2^{r-3} + \cdots + 2^{s-1} + 2^t - 1 + (2^{s-1} - j)}{2(2^{s-1} - j)} \end{aligned}$$

and by Lemma 2.3

$$= \binom{2^s + 2^{s-1} + 2^t - 1 + (2^{s-1} - j)}{2(2^{s-1} - j)}.$$

C vanishes for $j = 0$, so we may assume $j > 0$. Since $2^s - 2j < 2^s$, we see by Lemma 2.11, that $C = \binom{2^{s-1}+2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ is nonzero if and only if $j = 2^t$ or $j = 2^x + 2^y - 2^t$, where $t+1 \le x \le s-1$ and $0 \le y \le t$.
Case $k = 2^l$ $(0 \le l \le t-1,\ 0 \le j \le 2^l - 1)$: We have $C = \binom{2^{r-1}+2^t-2^l-1-j}{2^s-2j}$ which is zero for $j = 0$ and we only consider $j > 0$. Then we have

$$C = \binom{2^{r-1} - 2^{s-1} + 2^t - 2^l - 1 + (2^{s-1} - j)}{2(2^{s-1} - j)}.$$

Subcase when $s \ge t+2$: We have $j \le 2^l - 1 \le 2^{t-1} - 1 \le 2^{s-3} - 1$ and $2^{s-1} - j \ge 2^{s-2} + 2^{s-3} + 1$. Therefore by Lemma 2.3, we have

$$\begin{aligned} C &= \binom{2^{r-2} + 2^{r-3} + \cdots + 2^{s-1} + 2^t - 2^l - 1 + (2^{s-1} - j)}{2(2^{s-1} - j)} \\ &= \binom{2^{s-1} + 2^t - 2^l - 1 + (2^{s-1} - j)}{2(2^{s-1}) - j)} = \binom{2^s + (2^t - 2^l - 1 - j)}{2(2^{s-1}) - j)}. \end{aligned}$$

Since $j \le 2^l - 1$, we have $2^t - 2^l - 1 - j \ge 2^t - 2^{l+1} \ge 0$. Thus by Lemma 2.3, we have $C = \binom{2^t-2^l-1-j}{2^s-2j}$. If C is nonzero, we should have $2^s - 2j \le 2^t - 2^l - 1 - j$. But this is impossible because $j \ge 2^s - 2^t + 2^l + 1 > 2^s - 2^t \ge 2^t$ contradicts $j \le 2^l - 1$.
Subcase $s = t+1$: By Lemma 2.13, $C = \binom{2^{r-1}-2^l-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ is nonzero if and only if $j = 2^{s-1} - 2^l$ or $j = 2^{s-1} - (2^l - 2^\lambda)$ where $0 \le \lambda \le l$. This is impossible since $j \le 2^l - 1$. Summing up, we have $C \neq 0$ if and only if $k = 0$ and $j = 2^t$ or $j = 2^x + 2^y - 2^t$ where $0 \le y \le t$ and $t+1 \le x \le s-1$.

We want the condition for B to be nonzero. When $k = 0$ and $j = 2^t$, we have $B = \binom{2^{r-1}+2^t-1}{i}$ which is nonzero if either $0 \le i \le 2^t - 1$ or $2^{r-1} \le i \le 2^{r-1} + 2^t - 1$. When $k = 0$ and $j = 2^x + 2^y - 2^t$, we shoud have

$$B = \binom{2^{r-1} + 2^x + 2^y - 2^t - 1}{i} \neq 0. \tag{3.2}$$

Finally we consider the coefficient A.
Case $k = 0$, $j = 2^t$ and $0 \le i \le 2^t - 1$: This case conflicts with the allowability condition (3.1).
Case $k = 0$, $j = 2^t$ and $2^{r-1} \le i \le 2^{r-1} + 2^t - 1$: Clearly we have $A \neq 0$ if and only if $i = 2^{r-1}$.
Case $k = 0$, $j = 2^x + 2^y - 2^t$ where $0 \le y \le t$ and $t + 1 \le x \le s - 1$ with condition (3.2): Let $A = \binom{2^s+2^{t+1}-2^x-2^y-1+(2^{r-1}-i)}{2(2^{r-1}-i)}$ be nonzero. From $2^s + 2^{t+1} - 2^x - 2^y - 1 + (2^{r-1} - i) \ge 2(2^{r-1} - i)$, we have $i \ge 2^{r-1} - 2^s + (2^x + 2^y - 2^t - 1) - 2^t + 2$. Since $2^x + 2^y - 2^t - 1 \ge 2^t$, we have $i \ge 2^{r-1} - 2^s + 2 \ge 2^s + 2$. But for such i, (3.2) does not hold unless $i = 2^{r-1}$. Thus we have proven that a Kervaire dual appears if and only if $k = 0$, $i = 2^{r-1}$ and either $j = 2^t, (a, b) = (2^r + 2^s - 1, 2^r + 2^t - 1)$ or $j = 2^x + 2^y - 2^t$, $(a, b) = (2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1, 2^r - 2^t + 2^x + 2^y - 1)$, where $t + 1 \le x \le s - 1$, $0 \le y \le t$. □

Lemma 3.3. *Let $r \ge s+2$. There are no occurrences of Kervaire duals for $m = r - 2$.*

Proof. Suppose that there exists an occurrence of a Kervaire dual. Then we have $b - i - j - k = 2^{r-2} - 1$. $F = \binom{2^{r-2}-1}{k}$ is nonzero for all k with $0 \le k \le 2^{r-2} - 1$. We next consider $E = \binom{2^r-2^{r-2}-1-k}{2^t-2k}$. When $t = 0$, $E \neq 0$ if and only if $k = 0$ since $r - 2 > 0$. When $t \ge 1$, E is also nonzero for $k = 0$. Hereafter we assume that $k > 0$. Then we have

$$\begin{aligned} E &= \binom{2^{r-1} + 2^{r-2} - 2^{t-1} - 1 + (2^{t-1} - k)}{2(2^{t-1} - k)} \\ &= \binom{2^{r-1} + 2^{r-3} + 2^{r-4} + \cdots + 2^{t-1} - (2^{t-1} - k)}{2(2^{t-1} - k)}. \end{aligned}$$

We can remove higher 2-powers by Lemma 2.3 since $2^t > 2(2^{t-1} - k)$ and obtain $E = \binom{2^{t-1}-1+(2^{t-1}-k)}{2(2^{t-1}-k)}$. This is nonzero if and only if $k = 2^l$ where $0 \le l \le t-1$. Thus we have shown that $E \neq 0$ if and only if $k = 0$ or $k = 2^l$ $(0 \le l \le t-1)$. It is easy to see that $D = \binom{2^{r-2}+k-1}{j}$ is nonzero if and only if $0 \le j \le 2^{r-2} - 1$ when $k = 0$ and $0 \le j \le 2^l - 1$ or $2^{r-2} \le j \le 2^{r-2} + 2^l - 1$ $(0 \le l \le t - 1)$ when $k = 2^l$. We next consider $C = \binom{2^{r-1}+2^{r-2}+2^t-1-j-k}{2^s-2j}$.

Case $k=0$, $0 \le j \le 2^{r-2}-1$: When $j=0$, C is nonzero only if $s=r-2$. Hereafter we assume that $j>0$. Then by 2.3, we have

$$\begin{aligned} C &= \binom{2^{r-1}+2^{r-2}-2^{s-1}+2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)} \\ &= \binom{2^{r-1}+2^{r-3}+\cdots+2^{s-1}+2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)} \\ &= \binom{2^{s-1}+2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)}. \end{aligned}$$

Subcase $s=t+1$: $C=\binom{2^s-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ is nonzero if and only if $j=2^{s-1}=2^t$ by Lemma 2.6.
Subcase $s \ge t+2$: By Lemma 2.12, C is nonzero if and only if either $j=2^t$, $j=2^\mu-(2^t-2^\lambda)$ where $t+1 \le \mu \le s-1$, $0 \le \lambda \le t$ or $j=0$ (only when $r=s+2$).
Case $k=2^l$, $(0 \le l \le t-1)$, $0 \le j \le 2^l-1$: We have $C=\binom{2^{r-1}+2^{r-2}+2^t-2^l-1-j}{2^s-2j}$, which is nonzero for $j=0$ if and only if $r=s+2$. Assuming $j>0$ and by Lemma 2.3, we have

$$\begin{aligned} C &= \binom{2^{r-1}+2^{r-2}+\cdots+2^{s-1}+2^t-2^l-1+(2^{s-1}-j)}{2(2^{s-1}-j)} \\ &= \binom{2^{s-1}+2^t-2^l-1+(2^{s-1}-j)}{2(2^{s-1}-j)} = \binom{2^s+2^t-2^l-1-j}{2^s-2j}. \end{aligned}$$

Since $1 \le j \le 2^l-1$, we have $2^s \le 2^s+2^t-2^{l+1} \le 2^s+2^t-2^l-1-j$. Therefore by Lemma 2.3, we have $C=\binom{2^t-2^l-1-j}{2^s-2j}$, which is nonzero only if $2^t-2^l-1-j \ge 2^s-2j$, i.e. $j \ge 2^s-2^t+2^l+1$. This is incompatible with $1 \le j \le 2^l-1$. Thus we have shown that C is nonzero if and only if $k=2^l$ $(0 \le l \le t-1)$, $j=0$ and $r=s+2$.
Case $k=2^l$ $(0 \le l \le t-1)$ and $2^{r-2} \le j \le 2^{r-2}+2^l-1$: This case is incompatible with $0 \le j \le 2^{s-1}$ and is discarded.

So far we have shown that $CDEF$ is nonzero if and only if either

$$k=0, \quad j=\begin{cases} 2^t & \\ 2^\mu-(2^t-2^\lambda) & (t+1 \le \mu \le s-1,\ 0 \le \lambda \le t) \\ 0 & (\text{only when } s=r-2) \end{cases}$$

or

$$k=2^l \ \ (0 \le l \le t-1), \quad j=0 \ \ (\text{only when } s=r-2).$$

We turn to B. There are four cases:
Case $k=0$, $j=0$, $r=s+2$: We have $B=\binom{2^s-1}{i}$ which is nonzero for all i with $0 \le i \le 2^s-1=2^{r-2}-1$.

Case $k=0$, $j=2^t$: We have $B=\binom{2^{r-2}+2^t-1}{i}$ which is nonzero for $0\le i\le 2^t-1$ or $2^{r-2}\le i\le 2^{r-2}+2^t-1$.
Case $k=0$, $j=2^x+2^y-2^t$ $(t+1\le x\le s-1,\ 0\le y\le t)$: We should have

$$B=\binom{2^{r-2}+2^x+2^y-2^t-1}{i}\neq 0 \tag{3.3}$$

Case $k=2^l$, $j=0$ (only when $r=s+2$) : We have nonzero $B=\binom{2^{r-2}+2^l-1}{i}$ if and only if $0\le i\le 2^l-1$ or $2^{r-2}\le i\le 2^{r-2}+2^l-1$.

Finally we consider $A=\binom{2^{r-1}+2^{r-2}+2^s+2^t-1-i-j-k}{2^r-2i}$.
Case $r=s+2$, $k=j=0$, $0\le i\le 2^{r-2}-1$. This contradicts the allowability condition (3.1).
Case $k=0$, $j=2^t$, $0\le i\le 2^t-1$: In this case we have $i+j+k=2^{t+1}-1\le 2^{r-2}-1$. This does not satisfy allowability either.
Case $k=0$, $j=2^t$, $2^{r-2}\le i\le 2^{r-2}+2^t-1$: We have $i+j+k\le 2^{r-2}+2^{t+1}-1$. By allowability (3.1), we have

$$3(2^{r-2}+2^{t+1}-1)\ge 3(i+j+k)\ge 2^r+2^{r-2}+2^s+2^t+1.$$

This yields

$$2^{t+2}+2^t\ge 2^{r-1}+2^s+4\ge 2^{t+2}+2^{t+1}+4,$$

which is impossible.
Case $k=0$, $j=2^x+2^y-2^t$ $(t+1\le x\le s-1$ and $0\le y\le t)$ together with (3.3) : From (3.3), we have $i\le 2^{r-2}+2^x+2^y-2^t-1$ and we have $i+j+k\le 2^{r-2}+2^{x+1}+2^{y+1}-2^{t+1}-1$. From (3.1) we have

$$3(2^{r-2}+2^{x+1}+2^{y+1}-2^{t+1}-1)\ge 2^r+2^{r-2}+2^s+2^t+1.$$

This reduces to

$$2^{x+2}+2^{x+1}+2^{y+2}+2^{y+1}\ge 2^{r-1}+2^s+2^{t+2}+2^{t+1}+2^t+4.$$

This inequality together with

$$2^{s+1}+2^s+2^{t+2}+2^{t+1}\ge 2^{x+2}+2^{x+1}+2^{y+2}+2^{y+1}$$

yields the inequality $2^{s+1}\ge 2^{r-1}+2^t+4$ which is impossible since $r\ge s+2$.
Case $k=2^l$ $(0\le l\le t-1)$, $j=0$, $s=r-2$: If $0\le i\le 2^l-1$, we have $i+j+k\le 2^{l+1}-1\le 2^t-1$. This contradicts (3.1). Therefore we may assume that $2^{r-2}\le i\le 2^{r-2}+2^t-1$. Also by (3.1), we have $3(2^{r-2}+2^t-1)\ge 2^r+2^{r-2}+2^s+2^t+1$. This inequality is equivalent to $2^{t+1}\ge 2^{r-1}+2^s+4$ which is a contradiction.

Thus we have shown that the composition of coefficients $ABCDEF$ of Kervaire duals always vanish. □

Lemma 3.4. *IF $r\ge s+2$, there are no occurrences of Kervaire duals with $m=r-3$.*

Proof. Let $m = r-3$. By allowability (3.1), we have

$$3(i+j+k) \geq 2^r + 2^{r-1} + 2^{r-3} + 2^s + 2^t + 1.$$

From the condition $F \neq 0$, we have $0 \leq k \leq 2^{r-3}-1$. From the condition $C \neq 0$ we have $j \leq 2^{s-1}$. $B = \binom{2^{r-3}-1+j+k}{i}$ is nonzero only if

$$i \leq 2^{r-3} - 1 + (2^{r-3}-1) + 2^{s-1} = 2^{r-2} - 2 + 2^{s-1}.$$

Therefore we have

$$i+j+k \leq (2^{r-2} - 2 + 2^{s-1}) + (2^{r-3}-1) + 2^{s-1} = 2^{r-2} + 2^{r-3} + 2^s - 3.$$

From allowability (3.1),

$$3(2^{r-2} + 2^{r-3} + 2^s - 3) \geq 2^r + 2^{r-1} + 2^{r-3} + 2^s + 2^t + 1.$$

This gives $2^{s+1} \geq 2^{r-1} + 2^t + 10$, which is impossible. □

We have obtained the possible Kervaire dual occurrences in

$$Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$$

for $r \geq s+2$. The result is as follows:

Lemma 3.5. *Let $r \geq s+2$, then a Kervaire dual of dimension 2^r-2 occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ if and only if*

$$(a,b) = \begin{cases} (2^r + 2^s - 1, 2^r + 2^t - 1) \\ (2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1,\ 2^r - 2^t + 2^x + 2^y - 1) \\ \qquad (t+1 \leq x \leq s-1,\ 0 \leq y \leq t) \end{cases}$$

Next we consider the Kervaire dual occurrences when $r = s+1$ and $s \geq t+2$.

Lemma 3.6. *When $r = s+1$ and $s \geq t+2$, a Kervaire dual for $m = r-1$ occurs if and only if*

$$(a,b) = \begin{cases} (2^r + 2^s - 1,\ 2^r + 2^t - 1) \\ (2^r + 2^{r-1} + 2^{t+1} + 2^t - 2^l - 1,\ 2^{r-1} + 2^s - 2^{t+1} + 2^l - 1) \\ (2^r + 2^s + 2^t - 2^l - 1, 2^r + 2^l - 1) \\ (2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1,\ 2^r - 2^t + 2^x + 2^y - 1) \end{cases}$$

where $0 \leq l \leq t-1$, $0 \leq y \leq t$, and $t+1 \leq x \leq s-1$.

Proof. $F \neq 0$ if and only if $0 \leq k \leq 2^{r-1}-1$. $E = \binom{2^{r-1}-1-k}{2^t-2k}$ is nonzero for $k = 0$. When $t = 0$, E is nonzero only if $k = 0$. Hereafter we assume that $t > 0$. Clearly we should have $k \leq 2^{t-1}$ and we see that

$$E = \binom{2^{r-1} - 2^{t-1} - 1 + (2^{t-1} - k)}{2(2^{t-1} - k)}$$

is nonzero if and only if $k = 0$ or $k = 2^l$ $(0 \leq l \leq t-1)$ by Lemma 2.13. We shall study D under the condition $EF \neq 0$. If $k = 0$ we have $D = \binom{2^{r-1}-1}{j}$ which is nonzero for any j with $0 \leq j \leq 2^{r-1}-1$. When $k = 2^l$, then

$D = \binom{2^{r-1}+2^l-1}{j}$ is nonzero if either $0 \le j \le 2^l-1$ or $2^{r-1} \le j \le 2^{r-1}+2^l-1$. We next consider $C = \binom{2^{r-1}+2^t-1-j-k}{2^s-2j}$. If $C \neq 0$, then we have $j \le 2^{s-1}$.
Case $k = 0$ and $0 \le j \le 2^{s-1}$: We have $C = \binom{2^{s-1}+2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ which is nonzero if and only if

$$j = \begin{cases} 2^t \\ 0 \\ 2^x - (2^t - 2^y) \quad (t+1 \le x \le s-1,\ 0 \le y \le t). \end{cases}$$

by Lemma 2.12.
Case $k = 2^l$, $0 \le j \le 2^l - 1$ $(0 \le l \le t-1)$: $C = \binom{2^s+2^t-2^l-1-j}{2^s-2j}$ is nonzero for $j = 0$. For $1 \le j \le 2^l - 1$, since $2^s + 2^t - 2^l - 1 - j \ge 2^s$ and by Lemma 2.3, we have $C = \binom{2^t-2^l-1-j}{2^s-2j}$. C is nonzero only if $2^t - 2^l - 1 - j \ge 2^s - 2j$, which means $j \ge 2^s - 2^t + 2^l + 1$. This is impossible since $j \le 2^l - 1$. Thus we have shown that $CDE \neq 0$ if and only if either

$$k = 0, \quad j = \begin{cases} 2^t \\ 0 \\ 2^x + 2^y - 2^t \quad (t+1 \le x \le s-1,\ 0 \le y \le t) \end{cases}$$

or

$$k = 2^l \quad (0 \le l \le t-1), \quad j = 0.$$

We turn to

$$B = \binom{2^{r-1} - 1 + j + k}{i} \qquad \text{and}$$

$$A = \binom{2^{r-1} + 2^s + 2^t - 1 - i - j - k}{2^r - 2i} = \binom{2^{s+1} + 2^t - 1 - i - j - k}{2^{s+1} - 2i}.$$

The allowability condition (3.1) is

$$3(i + j + k) \ge 2^{r-1} + 2^s + 2^t + 1 \tag{3.4}$$

Case $k = 0$, $j = 2^t$: $B \neq 0$ if and only if $0 \le i \le 2^t - 1$ or $2^{r-1} \le i \le 2^{r-1} + 2^t - 1$. If $i \le 2^t - 1$, then we have $i + j + k \le 2^{t+1} - 1$ and this contradicts (3.4). So we may asuume that $2^{r-1} \le i \le 2^{r-1} + 2^t - 1$ and by considering A, we have $i = 2^{r-1}$.
Case $k = 0$, $j = 0$: $B = \binom{2^{r-1}-1}{i}$ is nonzero for $i \le 2^{r-1} - 1 = 2^s - 1$. For A we have

$$A = \binom{2^{s+1} + 2^t - 1 - i}{2^{s+1} - 2i} = \binom{2^s + 2^t - 1 + (2^s - i)}{2(2^s - i)},$$

which is, by Lemma 2.11, nonzero if and only if $i = 2^t$, $i = 0$ or $i = 2^x + 2^y - 2^t$ where $t + 1 \le x \le s$, $0 \le y \le t$. If $i = 2^t$ or $i = 0$, (3.4) is not satisfied. The allowability condition (3.4) for $i = 2^x + 2^y - 2^t$ is given by

$2^{x+1}+2^x+2^{y+1}+2^y \geq 2^{s+1}+2^{t+2}+1$. This is possible only for $x=s$ and we should have $i=2^s-2^t+2^y$ $(0 \leq y \leq t-1)$. Remark that we removed $l=t$ since $i \leq 2^s-1$.

Case $k=0$, $j=2^x+2^y-2^t$ where $t+1 \leq x \leq s-1$, $0 \leq y \leq t$: We assume that $B=\binom{2^{r-1}+2^x+2^y-2^t-1}{i}$ and A are nonzero. Then we have $i=2^{r-1}$ or $0 \leq i \leq 2^x+2^y-2^t-1$. When $0 \leq i \leq 2^x+2^y-2^t-1$, from (3.4) we have $2^{x+2}+2^{x+1}+2^{y+2}+2^{y+1} \geq 2^{s+1}+2^{t+2}+2^{t+1}+2^t+4$ which is satisfied only if $x=s-1$. So we divide this case into two subcases:

(a) $k=0$, $j=2^{s-1}+2^y-2^t$, $0 \leq i \leq 2^{s-1}+2^y-2^t-1$ $(0 \leq y \leq t)$

(b) $k=0$, $j=2^x+2^y-2^t$ $(t+1 \leq x \leq s-1,\ 0 \leq y \leq t)$, $i=2^{r-1}=2^s$.

Subcase (a) : If $A=\binom{2^s+2^{s-1}+2^{t+1}-2^y-1-i}{2^{s+1}-2i} \neq 0$, we have $2^s+2^{s-1}+2^{t+1}-2^y-1-i \geq 2^{s+1}-2i$, i.e. $i \geq 2^{s-1}-2^{t+1}+2^y+1$. If $B=\binom{2^{s-1}-2^t+2^y-1}{i} \neq 0$, we have $i \leq 2^{s-1}-2^t+2^y-1$. From these two inequalities, we have $2^t-2^y+1 \leq 2^{s-1}-i \leq 2^{t+1}-2^y-1 < 2^{s-1}$. By Lemma 2.3 we have $A=\binom{2^{t+1}-2^y-1+(2^{s-1}-i)}{2(2^{s-1}-i)}$. Therefore by Lemma 2.13, we have either $i=2^{s-1}-(2^{t+1}-2^\nu)-2^y$ or $i=2^{s-1}-(2^{t+1}-2^\mu)-(2^y-2^\lambda)$, where $y+2 \leq \nu \leq t+1$, $y+1 \leq \mu \leq t+1$, $0 \leq \lambda \leq y$. If $i=2^{s-1}-(2^{t+1}-2^\nu)-2^y$, then we find that $B=\binom{2^{s-1}-2^t+2^y-1}{2^{s-1}-2^{t+1}+2^\nu-2^y}$ vanishes by watching the 2^y components of B. Only the case $i=2^{s-1}-2^{t+1}+2^\mu-2^y+2^\lambda$ remains. If $y=t$ then $\mu=t+1$ and $i=2^{s-1}-2^t+2^\lambda$. If $\lambda=t$, this is impossible for then we have $i=2^{s-1}$. Therefore we have $i=2^{s-1}-2^t+2^l$ $(0 \leq l \leq t-1)$. If $y \leq t-1$, again by considering the 2^y components of B, we should have $\lambda=y$. Hence we have $i=2^{s-1}-2^{t+1}+2^\mu$ where $y+1 \leq \mu \leq t+1$. However $\mu=t+1$ is impossible since $i<2^{s-1}$. Similarly for B to be nonzero, the only possible value of μ is $\mu=t$. Thus we get $i=2^{s-1}-2^t$.

Subcase (b) : For all values of $k=0$, $i=2^s$, $j=2^x+2^y-2^t$, $ABCDEF$ is nonzero. This case may be summarized as follows:

$$\begin{cases} k=0, & j=2^{s-1}, \quad i=2^{s-1}-2^t+2^l \\ k=0, & j=2^{s-1}+2^l-2^t, \quad i=2^{s-1}-2^t \\ k=0, & j=2^x+2^y-2^t, \quad i=2^{r-1}=2^s \end{cases}$$

where $0 \leq l \leq t-1$, $t+1 \leq x \leq s-1$ and $0 \leq y \leq t$.

Case $k=2^l$ $(0 \leq l \leq t-1)$, $j=0$: We have $B=\binom{2^{r-1}+2^l-1}{i}$. If $i \leq 2^l-1$ then allowability (3.4) is not satisfied. Hence $i=2^s$ is the only value for i.

Here is a summary result of the possible values for $ABCDEF \neq 0$ in this proof.

$(k,\ j,\ i)$	(a,b)
$(0,2^t,2^{r-1})$	$(2^r+2^s-1, 2^r+2^t-1)$
$(0,0,2^s-2^t+2^l)$	$(2^r+2^s+2^{t+1}-2^l-1, 2^r-2^t+2^l-1)$
$(0,2^{s-1},2^{s-1}-2^t+2^l)$	$(2^r+2^s+2^{t+1}-2^l-1, 2^r-2^t+2^l-1)$
$(0,2^{s-1}+2^l-2^t,2^{s-1}-2^t)$	$(2^r+2^s+2^{t+1}+2^t-2^l-1, 2^r-2^{t+1}+2^l-1)$
$(0,2^x+2^y-2^t,2^{r-1})$	$(2^r+2^s+2^{t+1}-2^x-2^y-1, 2^r+2^x+2^y-2^t-1)$
$(2^l,0,2^{r-1})$	$(2^r+2^s+2^t-2^l-1, 2^r+2^l-1)$

$$(0 \le l \le t-1,\ t+1 \le x \le s-1,\ \ 0 \le y \le t)$$

Table 1

Remark that the admissible (a, b) for second and the third line in Table 1 coincide. This shows that the two occurrences of the Kervaire duals in for pairs (a, b) cancel out. □

Lemma 3.7. *Suppose that $r = s+1$ and $s \ge t+2$ and $m = r-2$. Then a Kervaire dual occurs if and only if*

$$(a,b) = \begin{cases} (2^r + 2^s + 2^t - 2^l - 1,\ 2^r + 2^l - 1) \\ (2^r + 2^s + 2^{t+1} + 2^t - 2^l - 1,\ 2^r - 2^{t+1} + 2^l - 1) \end{cases}$$

where $0 \le l \le t-1$.

Proof. Note that the allowability condition in the present case is

$$3(i+j+k) \ge 2^{s+1} + 2^s + 2^{s-1} + 2^t + 1.$$

F is nonzero if and only if $0 \le k \le 2^{r-2} - 1$. If $k = 0$ we have $E \ne 0$. When $k > 0$, $E = \binom{2^{r-1}+2^{r-2}-2^{t-1}-1+(2^{t-1}-k)}{2(2^{t-1}-k)} = \binom{2^{t-1}-1+(2^{t-1}-k)}{2(2^{t-1}-k)}$ and by Lemma 2.6, we have $k = 2^l$ $(0 \le l \le t-1)$. Therefore $EF \ne 0$ if and only if $k = 0$ or $k = 2^l$ where $0 \le l \le t-1$. We have $D = \binom{2^{r-2}-1+k}{j}$ and $C = \binom{2^{r-1}+2^{r-2}+2^t-1-j-k}{2^s-2j}$. We study the condition for $CDEF \ne 0$.
Case $k = 0$: If $j = 0$, C is nonzero. We assume that $j > 0$. Then we have $C = \binom{2^{s-1}+2^t-1-j}{2^s-2j} = \binom{2^t-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$, and by Lamma 2.6, C is nonzero if and only if $j = 2^{s-1} + 2^l - 2^t$ $(0 \le l \le t-1)$. Here we removed $l = t$ since $j \le 2^{r-2} - 1$.
Case $k = 2^l$ $(0 \le l \le t-1)$: We have

$$D = \binom{2^{s-1} + 2^l - 1}{j}, \quad C = \binom{2^s + 2^{s-1} + 2^t - 2^l - 1 - j}{2^s - 2j}.$$

This shows that j must satisfy $0 \le j \le 2^l - 1$ or $j = 2^{s-1}$. Remark that $0 \le j \le 2^l - 1$ is not a sufficient condition for $C \ne 0$.

Finally we want the condition for $ABCDEF \ne 0$.
Case $k = 0$, $j = 2^{s-1} + 2^l - 2^t$ where $0 \le l \le t-1$: We have

$$B = \binom{2^s - 2^t + 2^l - 1}{i}, \quad A = \binom{2^s + 2^{t+1} - 2^l - 1 + (2^s - i)}{2(2^s - i)}.$$

By Lemma 2.16, if $A \ne 0$, we have

$$i = \begin{cases} (1) & 2^\alpha - (2^{t+1} - 2^\beta) - (2^l - 2^\delta) \\ (2) & 0 \\ (3) & 2^{t+1} - (2^l - 2^\delta) \\ (4) & 2^{t+1} - 2^l \\ (5) & 2^\alpha - (2^{t+1} - 2^\gamma) - 2^l \end{cases}$$

where $t+2 \leq \alpha \leq s, l+1 \leq \beta \leq t+1, 0 \leq \delta \leq l, l+2 \leq \gamma \leq t+1$. We shall check each of the five cases closely.

First note that if $i \leq 2^{s-1}$, then the allowability condition is not satisfied. Therefore cases (2),(3) and (4) should be removed and in cases (1) and (5) we should limit the value of α as $\alpha = s$.

(1) : We can easily see that B vanishes unless $\delta = l$ by Lemma 2.1. Then we also see that $B = \binom{2^s-2^t+2^l-1}{2^s-2^{t+1}+2^\beta}$ is zero unless $\beta = t$. These observations show that $i = 2^s - 2^t$. The allowability condition is satisfied only for $i = 2^s - 2^t$. Cases (2), (3) and (4) do not satisfy allowability and should be eliminated.

(5) : We have $i = 2^s - 2^{t+1} + 2^\gamma - 2^l$ where $l+2 \leq \gamma \leq t+1$. In this case we have $B = 0$. Thus we have shown that in this case $ABCDEF \neq 0$ if and only if $k = 0$, $j = 2^{s-1} + 2^l - 2^t$ and $i = 2^s - 2^t$.

Case $k = 2^l$, $j = 2^{s-1}$ where $0 \leq l \leq t-1$: We have

$$B = \binom{2^s + 2^l - 1}{i}, \quad A = \binom{2^s + 2^t - 2^l - 1 + (2^s - i)}{2(2^s - i)}.$$

If $0 \leq i \leq 2^l - 1$ the allowability condition is not satisfied. Therefore $ABCDEF \neq 0$ if and only if $i = 2^s$. We summarize the result of the proof.

$(k,\ j,\ i)$	$(a,\ b)$
$(0, 2^{s-1} + 2^l - 2^t, 2^s - 2^t)$	$(2^r + 2^s + 2^{t+1} + 2^t - 2^l - 1, 2^r - 2^{t+1} + 2^l - 1)$
$(2^l, 2^{s-1}, 2^s)$	$(2^r + 2^s + 2^t - 2^l - 1, 2^r + 2^l - 1)$
$(0 \leq l \leq t-1)$	

Table 2

□

Lemma 3.8. *If $r = s+1$, $s \geq t+2$ and $m = r-3$, there are no occurrences of Kervaire duals.*

Proof. We see that $E = \binom{2^{s+1}-2^{s-2}-1-k}{2^t-2k}$ is nonzero for $k = 0$ if $s \geq t+3$ and is zero if $s = t+2$ by checking the 2^t components of E. For $k > 0$, by Lemma 2.3, we have $E = \binom{2^{t-1}+(2^{t-1}-k)}{2(2^{t-1}-k)}$. Hence $E \neq 0$ if and only if $k = 2^l$ $(0 \leq l \leq t-1)$. $D = \binom{2^{s-2}-1+k}{j}$ is nonzero only if $j \leq 2^{s-2} - 1 + k \leq 2^{s-2} + 2^{t-1} - 1$, and $B = \binom{2^{s-2}-1+j+k}{i}$ is nonzero only if $i \leq 2^{s-2} - 1 + j + k \leq 2^{s-1} + 2^t - 2$. Therefore we have $i + j + k \leq 2^{s-1} + 2^{s-2} + 2^{t+1} - 3$, which is incompatible with the allowability. □

We here summarize the result for $r = s+1$ and $s \geq t+2$ by Lemmas 3.6,3.7 and 3.8:

Lemma 3.9. *If $r = s+1$ and $s \geq t+2$, a Kervaire dual of dimension $2^r - 2$ occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ if and only if*

$$(a,b) = \begin{cases} (2^r + 2^s - 1, 2^r + 2^t - 1) \\ (2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1, 2^r - 2^t + 2^x + 2^y - 1) \end{cases}$$

where $t+1 \leq x \leq s-1$, $0 \leq y \leq t$.

Finally we deal with the case when $r = s+1$ and $s = t+1$. Although this case is a special case of the result of [2], we shall give its proof here.

Lemma 3.10. *If $r = s+1$ and $s = t+1$, a Kervaire dual occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ for $m = r-1$, if and only if*

$$(a,b) = \begin{cases} (2^r + 2^s - 1, 2^r + 2^t - 1) \\ (2^r + 2^s + 2^t - 2^l - 1, 2^r + 2^l - 1) \qquad (0 \le l \le t-1). \end{cases}$$

Proof. First we note that the allowability condition is $3(i+j+k) \ge 2^{s+1} + 2^{s-1} + 1$. Considering $F = \binom{2^s-1}{k}$, we have $0 \le k \le 2^s - 1$. Next consider the case when $s = 1$, and we have $E \ne 0$ if and only if $k = 0$. If $s \ge 2$, we have $E = \binom{2^{s-1}+2^{s-2}-1+(2^{s-2}-k)}{2(2^{s-2}-k)}$. If $k = 0$, we have $E \ne 0$. For $k > 0$ we have $k = 2^l$ where $0 \le l \le t-1 = s-2$. Hence $EF \ne 0$ if and only if either $k = 0$ or $k = 2^l$ where $0 \le l \le s-2$.

As to D, if $k = 0$, then $D = \binom{2^s-1}{j}$ is nonzero if and only if $0 \le j \le 2^s - 1$. If $k = 2^l$, then $D = \binom{2^s+2^l-1}{j}$ is nonzero if and only if $0 \le l \le 2^l - 1$ or $2^s \le j \le 2^s + 2^l - 1$.

Next we go to $C = \binom{2^s+2^{s-1}-1-j-k}{2^s-2j}$.

Case $k = 0$, $0 \le j \le 2^s - 1$: We have $C = \binom{2^s-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ and C is nonzero if and only if $j = 0$ and $j = 2^{s-1}$.

Case $k = 2^l$, $0 \le j \le 2^l - 1$ $(0 \le l \le s-2)$: We have $C = \binom{2^s-2^l-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ and by Lemma 2.13, we find that C is nonzero if and only if $j = 0$.

We shall check the non-vanishing condition for

$$B = \binom{2^s - 1 + j + k}{i} \quad \text{and} \quad A = \binom{2^{s+1} + 2^{s-1} - 1 - i - j - k}{2^{s+1} - 2i}.$$

Case $k = 0$, $j = 0$: If $B \ne 0$, then $0 \le i \le 2^s - 1$ holds. From $A = \binom{2^s+2^{s-1}-1+(2^s-j)}{2(2^s-i)}$, we have $i = 2^{s-1}$, $i = 0$ or $i = 2^\lambda$ where $0 \le \lambda \le s-1$. However in each case, the allowability condition fails.

Case $k = 0$, $j = 2^{s-1}$: $B = \binom{2^s+2^{s-1}-1}{i}$ is nonzero only if $0 \le i \le 2^{s-1} - 1$ or $i = 2^s$. $A = \binom{2^s-1+(2^s-i)}{2(2^s-i)}$ is nonzero if and only if $i = 2^\kappa$ for $0 \le \kappa \le s$. If $0 \le i \le 2^{s-1} - 1$, then we have $i = 2^l$ for $0 \le l \le s-2$. This does not satisfy the allowability. Thus there is no choice for i but $i = 2^s$. In this case, $k = 0$, $j = 2^{s-1}$, $i = 2^s$ and $(a,b) = (2^r + 2^s - 1, 2^r + 2^t - 1)$.

Case $k = 2^l$, $j = 0$ $(0 \le l \le s-2)$: We have $B = \binom{2^s+2^l-1}{i}$. If $0 \le i \le 2^l - 1$, then the allowability condition is not satisfied. Thus we have $i = 2^s$, and the corresponding (a,b) is $(2^r+2^s+2^t-2^l-1, 2^r+2^l-1)$ where $0 \le l \le s-2$. □

Lemma 3.11. *Suppose that $r = s+1$ and $s = t+1$. Then a Kervaire dual occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ for $m = r-2$, if and only if $(a,b) = (2^r + 2^s + 2^t - 2^l - 1, 2^r + 2^l - 1)$ where $0 \le l \le t-1$.*

Proof. The allowability condition is $3(i+j+k) \geq 2^{s+2}+1$. From $F \neq 0$, we have $0 \leq k \leq 2^{s-1}-1$. If $s=1$, we find that $E = \binom{2-k}{1-2k}$ is always zero. So we may assume that $s > 1$. Since $E = \binom{2^s+2^{s-1}-1-k}{2^{s-1}-2k}$ is zero for $k=0$, we assume that $k \geq 1$. Then E is nonzero if and only if $k = 2^l$ $(0 \leq l \leq s-2)$. We have $D = \binom{2^{s-1}+2^l-1}{j}$, which is nonzero for $0 \leq j \leq 2^l-1$ or $2^{s-1} \leq j \leq 2^{s-1}+2^l-1$. $C = \binom{2^{s+1}-2^l-1-j}{2^s-2j}$ is nonzero for $j=0$, but $j=0$ does not satisfy the allowability. Let $j > 0$, then by removing the higher 2-powers, we see that $C = \binom{2^{s-1}-2^l-1+(2^{s-1}-j)}{2(2^{s-1}-j)}$ is nonzero for $j = 2^\beta - 2^l$ or $j = 2^\alpha - (2^l - 2^\lambda)$ where $l+2 \leq \beta \leq s-1$, $l+1 \leq \alpha \leq s-1$ and $0 \leq \lambda \leq l$. All these values do not satisfy $j \leq 2^l-1$ and should be dismissed. The only remaining case is $j = 2^{s-1}$. $B = \binom{2^s+2^l-1}{i}$ is nonzero for $0 \leq i \leq 2^l-1$ or $i = 2^s$. But $0 \leq i \leq 2^l-1$ does not satisfy the allowability. Therefore we have $i = 2^s$ and we get the assertion. □

Lemma 3.12. *If $r = s+1$ and $s = t+1$, then there are no occurrences of Kervaire duals for $m = r-3$.*

Proof. The allowability condition is $3(i+j+k) \geq 2^{s+2}+2^{s-1}+2^{s-2}+1$. From $F \neq 0$, we have $0 \leq k \leq 2^{s-2}-1$. By Lemma 2.3 we have

$$E = \binom{2^s+2^{s-1}-1+(2^{s-2}-k)}{2(2^{s-2}-k)} = \binom{2^{s-1}-1+(2^{s-2}-k)}{2(2^{s-2}-k)},$$

and $E \neq 0$ only for $k=0$ since $k \leq 2^{s-2}-1$. Then $D = \binom{2^{s-2}-1}{j}$ is nonzero for $0 \leq j \leq 2^{s-2}-1$. If $B = \binom{2^{s-2}-1-j}{i} \neq 0$, we have $i \leq 2^{s-1}-2$. These conditions do not satisfy the allowability. □

Lemma 3.13. *If $r = s+1$ and $s = t+1$, then there are no occurrences of Kervaire duals for $m = r-4$.*

Proof. Considering F we have $k \leq 2^{s-3}-1$. From $E = \binom{2^{s-3}-1+(2^{s-2}-k)}{2(2^{s-2}-k)} \neq 0$ we have $k=0$ or $k = 2^{s-3}+2^\alpha$ where $0 \leq \alpha \leq s-3$. This is impossible since $k < 2^{s-3}$. □

We summarize the case when $r = s+1$ and $s = t+1$.

Lemma 3.14. *If $r = s+1$ and $s = t+1$, a Kervaire dual of dimension 2^r-2 occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ if and only if $i = (2^r+2^s-1, 2^r+2^t-1)$.*

The results of Lemmas 3.5, 3.9 and 3.14 show

Lemma 3.15. *Let $r > s > t \geq 0$ and $0 < b \leq a \leq 2b$, then a Kervaire dual of dimension 2^r-2 occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^r}_* u(a,b)$ if and only if $(a,b) = (2^r+2^s-1, 2^r+2^t-1)$ or $(2^r+2^s+2^{t+1}-2^x-2^y-1, 2^r-2^t+2^x+2^y-1)$ $(t+1 \leq x \leq s-1, 0 \leq y \leq t)$.*

We now go to the right hand side of our main theorm. Let $0 < b \leq a \leq 2b$ and consider

$$Sq_*^{2^t} Sq_*^{2^s} u(a,b) = \sum_{i,j} A'B'C'D'u(c',d'), \tag{3.5}$$

where

$$A' = \binom{a-2^s}{2^s-2i}, \qquad B' = \binom{b-i}{i},$$
$$C' = \binom{a-2^s-2^t+i}{2^t-2j}, \qquad D' = \binom{b-i-j}{j},$$
$$c' = a-2^s-2^t+i+j, \qquad d' = b-i-j$$

We look for the condition for (a,b) when Kervaire duals with $d' = 2^k - 1$ occur. That is, we want to find (a,b) such that $A'B'C'D'$ is nonzero for some $k \leq r$.

Lemma 3.16. *Let $r > s > t \geq 0$ and $0 < b \leq a \leq 2b$. Then a Kervaire dual of dimension $2^{r+1}-2$ occurs in $Sq_*^{2^t} Sq_*^{2^s} u(a,b)$ if and only if*

$$(a,b) = \begin{cases} (2^r+2^s-1, 2^r+2^t-1) \\ (2^r+2^s+2^{t+1}-2^x-2^y-1, 2^r-2^t+2^x+2^y-1), \end{cases}$$

where $t+1 \leq x \leq s-1$, $0 \leq y \leq t$.

Proof. We note that when $d' = b-i-j = 2^k-1$ $(k \leq r)$, we have

$$A' = \binom{a-2^s}{2^s-2i} = \binom{2^{r+1}+2^t-2^k-1-i-j}{2^s-2i}$$
$$B' = \binom{b-i}{i} = \binom{2^k-1+j}{i}$$
$$C' = \binom{2^{r+1}-2-(b-i)}{2^t-2j} = \binom{2^{r+1}-2^k-1-j}{2^t-2j}$$
$$D' = \binom{2^k-1}{j}$$

with allowability condition $3(i+j) \geq 2^{r+1}+2^s+2^t-(2^{k+1}+2^k)+1$.

First note that if $k = r$, the allowability condition is always satisfied. However if $k \leq r-1$, $A'C' \neq 0$ implies $i+j \leq 2^{s-1}+2^{t-1}$ which fails to satisfy the allowability. Therefore we have only to consider the case $k = r$. Then we have $b-i-j = 2^r-1$ and from $D' \neq 0$, we have $0 \leq j \leq 2^r-1$. Thus we get $C' = \binom{2^r-1-j}{2^t-2j}$. If $t = 0$, $C' \neq 0$ if and only if $j = 0$. If $t \geq 1$, then we have $C' = \binom{2^r-2^{t-1}-1+(2^{t-1}-j)}{2(2^{t-1}-j)}$ and by Lemma 2.13 we have $j = 0$ or $j = 2^l$ where $0 \leq l \leq t-1$. We next consider $B' = \binom{2^r-1+j}{i}$ and $A' = \binom{2^r+2^t-1-i-j}{2^s-2i}$.

Case $j = 0$: From $B' \neq 0$ we have $0 \leq i \leq 2^r - 1$ and $A' = \binom{2^r - 2^{s-1} + 2^t - 1 + (2^{s-1} - i)}{2(2^{s-1} - i)}$. If $i = 0$, then we have $A' = 0$. So we may assume that $i > 0$. Then by deleting higher 2-powers we obtain $A' = \binom{2^{s-1} + 2^t - 1 + (2^{s-1} - i)}{2(2^{s-1} - i)}$. If $s \geq t + 2$, we have, by Lemma 2.11, $i = 2^t$ or $i = 2^x + 2^y - 2^t$ where $t + 1 \leq x \leq s - 1$, $0 \leq y \leq t$. On the other hand, if $s = t + 1$, from $A' = \binom{2^s - 1 + (2^{s-1} - i)}{2(2^{s-1} - i)} \neq 0$ we have $i = 2^{s-1} = 2^t$.
Case $j = 2^l$ where $0 \leq l \leq t - 1$: If $A' = \binom{2^r + 2^t - 2^l - 1 - i}{2^s - 2i}$ is nonzero, we have $i \leq 2^{s-1}$ and $i \neq 0$. Therefore $B' = \binom{2^r + 2^l - 1}{i}$ is nonzero for $0 < i \leq 2^l - 1$. From the expression $A' = \binom{2^{s-1} + 2^t - 2^l - 1 + (2^{s-1} - i)}{2(2^{s-1} - i)}$ and by Lemma 2.16, we have the following possibilities for i:

$$i = \begin{cases} 2^\alpha - (2^t - 2^\beta) - (2^l - 2^\delta) \\ 2^t - (2^l - 2^\delta) \\ 2^t - 2^l \\ 2^\alpha - (2^t - 2^\epsilon) - 2^l \end{cases}$$

where $t + 1 \leq \alpha \leq s - 1$, $l + 1 \leq \beta \leq t$, $0 \leq \delta \leq l$, $l + 2 \leq \epsilon \leq t$. All cases contradicts $i \leq 2^l - 1$. Thus the case $j = 2^l$ should be deleted. And the proof is completed. □

Proof of Theorem 1.1:

Lemma 3.15 shows that a Kervaire dual of dimension $2^r - 2$ occurs in $Sq^{2^t}_* Sq^{2^s}_* Sq^{2^t}_* u(a, b)$ if and only if $(a, b) = (2^r + 2^s - 1, 2^r + 2^t - 1)$ or $(2^r + 2^s + 2^{t+1} - 2^x - 2^y - 1, 2^r - 2^t + 2^x + 2^y - 1)$ where $t + 1 \leq x \leq s - 1$, $0 \leq y \leq t$. Of course the latter case should be removed if $s = t + 1$. According to Lemma 3.16, this coincides with the case when a Kervaire dual of dimension $2^{r+1} - 2$ occurs in $Sq^{2^s}_* Sq^{2^t}_* u(a, b)$. Thus the proof of our main theorem is completed.

References

[1] C. H. Giffen, Smooth homotopy projective spaces, Bull. Amer. Math. Soc., 75(1969), 509–513.

[2] Y. Kitada, Relations among smooth Kervaire classes and smooth involutions on homotopy spheres, Kodai Math. Jour., 11(1988), 387–402.

[3] I. Madsen, On the action of the Dyer-Lashof algebra in $H_*(G)$, Pac. J. Math., 60(195), 235–275.

[4] I. Madsen and R. J. Milgram, The classifying spaces for surgery and cobordism of manifolds, Annals of Mathematics Studies, No. 92, Princeton Univ. Press. 1979.

[5] R. J. Milgram, The mod 2 sphereical characteristic clases, Ann. of Math., 92(1970), 238–261.

[6] G. Nishida, Cohomology operations in iterated loop spaces, Proc. Japan Acad. 44 (1968), 104–109.

[7] N. E. Steenrod and D. B. A. Epstein, Cohomology operations, Annals of Mathematics Studies, No. 50, Princeton Univ. Press. 1962.

Division of Electrical and Computer Engineering, Faculty of Engineering,
Yokohama National University, Hodogaya-ku, Yokohama 240-8501 Japan
E-mail address: kitada@mathlab.sci.ynu.ac.jp

プログラム

```
;;
;;  Calculation of Kervaire classes
;;
;;  (C) 2000, Yasuhiko Kitada
;;  Yokohama National University
;;

;;
;;(require-library "compat.ss")
;;
(define (ld) (load "sqdl.scm"))
(require 'sort)
;;
;; constants
;;
(define **max-dim** 200)
(define **adem-max-dim** 100)
(define **orig-heapsize** 5000)
;;
(define *derived-zero-functions*
  (make-vector (+ **max-dim** 1) #f))
(define *zero-functions*
  (make-vector (+ **max-dim** 1) #f))
(define *generating-zero-functions*
  (make-vector (+ **max-dim** 1) #f))
(define *sq-table*
  (make-vector (+ **max-dim** 1) #t))

(do ((i 0 (+ i 1)))
    ((> i 8))
  (vector-set! *zero-functions* i  '())
  (vector-set! *derived-zero-functions* i '())
  (vector-set! *generating-zero-functions* i '()))
;;(vector-set! *zero-functions* 9 '(((6 (2 1)) (2 (4 2 1)))))
;;
;; Misc
;;

;;
;; Ex. (pop-push '(1 2 3) '(a b)) => (3 2 1 a b)
;;
(define (pop-push a b)
  (if (null? a)
      b
```

```
      (pop-push (cdr a) (cons (car a) b))))

;;
;; Ex. (delete-nth 0 '(1 2 3)) => (2 3)
;;     (delete-nth 3 '(1 2 3)) => (1 2 3)
;;
(define (delete-nth n lst)
  (define (iter stk k rem)
    (if (null? rem)
        lst
        (if (zero? k)
            (pop-push stk (cdr rem))
            (iter (cons (car rem) stk)
                  (- k 1)
                  (cdr rem)))))
  (iter '() n lst))

;;
;; Ex. (replace-nth 1 '(1 2 3) 'a) => (1 a 3)
;;     (replace-nth 3 '(a b c) 'x) => (a b c)
;;
(define (replace-nth n lst new-obj)
  (define (iter stk k rem)
    (if (zero? k)
        (pop-push stk (cons new-obj (cdr rem)))
        (iter (cons (car rem) stk) (- k 1) (cdr rem))))
  (if (>= n (length lst))
      lst
      (iter '() n lst)))

(define (remove-if pred? lst)
  (define (iter rem res)
    (if (null? rem)
        (reverse res)
        (if (pred? (car rem))
            (iter (cdr rem) res)
            (iter (cdr rem) (cons (car rem) res)))))
  (iter lst '()))

(define (preserve-if pred? lst)
  (define (iter rem res)
    (if (null? rem)
        (reverse res)
        (if (pred? (car rem))
```

```
            (iter (cdr rem) (cons (car rem) res))
            (iter (cdr rem) res))))
  (iter lst '()))

(define (max-list-length lstlst)
  (define (iter rem m)
    (if (null? rem)
        m
        (let ((m1 (length (car rem))))
          (iter (cdr rem)
                (if (> m1 m)
                    m1
                    m)))))
  (iter lstlst 0))

;;
(define (sorted-member obj lst leq?)
  (cond ((null? lst) #f)
        ((equal? obj (car lst)) lst)
        ((leq? (car lst) obj)
         (sorted-member obj (cdr lst) leq?))
        (else #f)))

(define (2power-minus-one? n)
  (sorted-member n '(1 3 7 15 31 63 127 255 511 1023 2047) <=))

;;
;; Binomial coefficient reduced modulo 2
;;
(define (binom2 n m)
  (cond ((or (< n 0) (< m 0) (< n m)) 0)
        ((or (zero? m) (= n m)) 1)
        ((and (even? n) (odd? m)) 0)
        (else (binom2 (quotient n 2) (quotient m 2)))))

;;
;; global variables for heap sort
;;
(define *orig-heapsize* **orig-heapsize**)
(define *heapsize* *orig-heapsize*)
(define *heap* (make-vector (+ *heapsize* 1)))
(define *cnt* 0)
(define *heavier?* <)
```

```
;;
;; Heap sort
;;
(define (upward-check! i)
  (let ((up (quotient i 2))
        (curr (vector-ref *heap* i)))
    (if (> i 1)  ;; not root
        (if (*heavier?*
             (vector-ref *heap* up)
             curr)
            (begin
              (vector-set! *heap*
                           i
                           (vector-ref *heap* up))
              (vector-set! *heap* up curr)
              (upward-check! up))))))

(define (downward-check! i)
  (let ((left (* i 2))
        (right (+ (* i 2) 1))
        (curr (vector-ref *heap* i)))
    (cond ((<= right *cnt*)
           (if (*heavier?* (vector-ref *heap* right)
                           (vector-ref *heap* left))
               (if (*heavier?* curr
                               (vector-ref *heap* left))
                   (begin
                     (vector-set! *heap*
                                  i
                                  (vector-ref *heap* left))
                     (vector-set! *heap* left curr)
                     (downward-check! left)))
               (if (*heavier?* curr
                               (vector-ref *heap* right))
                   (begin
                     (vector-set! *heap*
                                  i
                                  (vector-ref *heap* right))
                     (vector-set! *heap* right curr)
                     (downward-check! right)))))
          ((= *cnt* left)
           (if (*heavier?* curr (vector-ref *heap* left))
               (begin
```

```
                (vector-set! *heap*
                             i
                             (vector-ref *heap* left))
                (vector-set! *heap* left curr)))))))

(define (push-heap obj)
  (if (<= *heapsize* *cnt*)
      (begin
        (let* ((new-heapsize (* *heapsize* 2))
               (new-heap (make-vector (+ new-heapsize 1))))
          (do ((i 0 (+ i 1)))
              ((> i *heapsize*))
            (vector-set! new-heap
                         i
                         (vector-ref *heap* i)))
          (set! *heapsize* new-heapsize)
          (set! *heap* new-heap))))
  (set! *cnt* (+ *cnt* 1))
  (vector-set! *heap* *cnt* obj)
  (upward-check! *cnt*))

(define (pop-heap)
  (if (> *cnt* 0)
      (let ((lightest (vector-ref *heap* 1)))
        (vector-set! *heap*
                     1
                     (vector-ref *heap* *cnt*))
        (set! *cnt* (- *cnt* 1))
        (downward-check! 1)
        lightest)
      '*empty-heap*))

(define (push-list-to-heap lst)
  (do ((rem lst (cdr rem)))
      ((null? rem))
    (push-heap (car rem))))

(define (full-pop-heap)
  (do ((res '() (cons (pop-heap) res)))
      ((zero? *cnt*)
       (begin (set! *cnt* 0)
              res))))

(define (full-pop-heap-mod2)
```

```
  (define (iter res)
    (cond ((zero? *cnt*) res)
          ((null? res)
           (iter (cons (pop-heap) res)))
          (else (let ((top (pop-heap)))
                  (if (equal? top (car res))
                      (iter (cdr res))
                      (iter (cons top res)))))))
  (iter '()))

;;
;; Global table for adem2
;;
(define *adem-max-dim* **adem-max-dim**)
(define *adem-table*
  (make-vector (+ *adem-max-dim* 1)))
(do ((i 1 (+ i 1)))
    ((> i *adem-max-dim*))
  (vector-set! *adem-table* i (make-vector (* 2 i) #f)))

;;
;; Adem relations in Steenrod algebra : Sq operations
;;
;;  Sq monomial = list of integers e.g. (8 3 1) for
;;     Sq^8 Sq^3 Sq^1
;;  Sq expressions = list of Sq monomials
;;  e.g. ((12)(11 1)(8 3 1)) for
;;     Sq^12 + Sq^11 Sq^1 + Sq^8 Sq^3 Sq^1
;;
(define *debug* #t)

;;
;; If the same two items appear in the list
;; they are removed.  If the same item appears
;; even times, they are all removed, whereas
;; if it appears odd times, only one item remains.
;;  e.g (remove-same-pairs '(1 1 3 2 1 3 2 2 1 3)) => (2 3)
;;
(define (remove-same-pairs lis)
  (define (iter res rem)
    (if (or (null? rem) (null? (cdr rem)))
        (pop-push res rem)
        (let ((i (find-pos (car rem) (cdr rem))))
          (if (null? i)
```

```
              (iter (cons (car rem) res)
                    (cdr rem))
              (iter res
                    (delete-nth i (cdr rem)))))))
  (if (null? lis)
      lis
      (iter '() lis)))
;;
;; Get the position of the item in the list
;;  (comparison by "equal?")
;;
;; Ex. (find-pos 'b '(a b c d)) => 1
;;     (find-pos 'x '(a b c d)) => ()
;;
(define (find-pos item lst)
  (define (iter i inp)
    (if (null? inp)
        '()
        (if (equal? item (car inp))
            i
            (iter (+ i 1) (cdr inp)))))
  (iter 0 lst))

(define (sqm>? a b)
  (define (same-length-iter aa bb)
    (cond ((null? aa) #f)
          ((< (car aa) (car bb)) #t)
          ((> (car aa) (car bb)) #f)
          (else (same-length-iter (cdr aa) (cdr bb)))))
  (let ((la (length a)) (lb (length b)))
    (cond ((> la lb) #t)
          ((< la lb) #f)
          (else (same-length-iter a b)))))

;;
;; Ex. (adem2 4 4) => ((6 2) (7 1))
;;  Sq^4 Sq^4 = sq^6 Sq&2 + Sq^7 Sq^1
;;

(define (adem2-aux a b)
  (define (iter stk i)
    (if (> (* i 2) a)
        (reverse stk)
        (if (zero? (binom2 (- b 1 i) (- a (* i 2))))
```

```
            (iter stk (+ i 1))
            (if (zero? i)
                (iter
                 (cons (list (+ a b)) stk)
                 (+ i 1))
                (iter
                 (cons (list (- (+ a b) i) i) stk)
                 (+ i 1))))))
  (if (>= a (+ b b))
      (list (list a b))
      (reverse (iter '() 0))))

(define (adem2 a b)
  (cond ((>= a (+ b b)) (list (list a b)))
        ((zero? a) (list (list b)))
        ((zero? b) (list (list a)))
        ((and (<= b *adem-max-dim*)
              (<= a (+ b b)))
         (let ((val (vector-ref
                     (vector-ref *adem-table* b)
                     a)))
           (if val
               val
               (let ((res (adem2-aux a b)))
                 (vector-set!
                  (vector-ref *adem-table* b)
                  a
                  res)
                 res))))
        (else (adem2-aux a b))))

(define (add-asqe asqe1 asqe2)
  (define (iter rem1 rem2 stk)
    (cond ((null? rem1)(pop-push stk rem2))
          ((null? rem2)(pop-push stk rem1))
          ((sqm>? (car rem1) (car rem2))
           (iter (cdr rem1) rem2 (cons (car rem1) stk)))
          ((sqm>? (car rem2) (car rem1))
           (iter rem1 (cdr rem2) (cons (car rem2) stk)))
          (else
           (iter (cdr rem1) (cdr rem2) stk))))
  (cond ((null? asqe1) asqe2)
        ((null? asqe2) asqe1)
        (else (iter asqe1 asqe2 '()))))
```

```
(define (add-asqe-list asqe-list)
  (define (iter rem res)
    (if (null? rem)
        res
        (iter (cdr rem)
              (add-asqe (car rem) res))))
  (iter asqe-list '()))
;;
;; cal-stack is a list of calculation status:
;; a calculation status is
;; (<rhead> <tail>) where <tail> is an admissible
;; sq monomial and <rhead> is reversed head part.
;; acc is a binary tree of sqm's.
;;
(set! *heavier?* sqm>?)
(set! *cnt* 0)

(define (sq-adem cal-stack)
  (cond ((null? cal-stack)
         (full-pop-heap-mod2))
        ((null? (caar cal-stack))
         (push-heap (cadar cal-stack))
         (sq-adem (cdr cal-stack)))
        (else
         (let ((rhd (caar cal-stack))
               (tl (cadar cal-stack)))
           (if (or (null? tl)
                   (>= (car rhd) (* 2 (car tl))))
               (sq-adem (cons (list (cdr rhd)
                                    (cons (car rhd) tl))
                              (cdr cal-stack)))
               (let ((res (adem2 (car rhd) (car tl))))
                 (if (null? res)
                     (sq-adem (cdr cal-stack))
                     (sq-adem
                      (append
                       (map
                        (lambda (x)
                          (list (pop-push x (cdr rhd))
                                (cdr tl)))
                        res)
                       (cdr cal-stack)))))))))))
```

```
;;
;; compatibility with previous version
;;
(define (adem-monom monom)
  (sq-adem (list (list (reverse monom) '()))))
;;
;; "Sq less or equal"
;; Ordering of Sq monomials
;; First priority is length
;;    (4 2) <= (3 2 1)
;; Second priority is lexical reverse order
;;    (4 3 1) <= (4 2 2)
;;
(define (sqleq a b)
  (let ((la (length a)) (lb (length b)))
    (cond ((null? a) #t)
          ((null? b) #f)
          ((> la lb) #t)
          ((< la lb) #f)
          ((< (car a) (car b)) #t)
          ((> (car a) (car b)) #f)
          (else (sqleq (cdr a) (cdr b))))))

(define sq
  (lambda x (sq-adem (list (list (reverse x) '())))))

(define sqadd add-asqe)

(define (sqmulmonos sqm1 sqm2)
  (sq-adem (list (list (reverse (append sqm1 sqm2)) '()))))

(define (sqmulone sqm sqe)
  (let ((rh (reverse sqm)))
    (sq-adem
     (map (lambda (x) (list rh x)) sqe))))

(define (sqmul sqe1 sqe2)
  (define (iter rem res)
    (if (null? rem)
        res
        (let ((rhd (reverse (car rem))))
          (iter
           (cdr rem)
           (add-asqe
```

```
            (sq-adem
             (map (lambda (tl)
                    (list rhd tl))
                  sqe2))
            res)))))
  (iter sqe1 '()))

(define (sqbra sqe1 sqe2)
  (sqadd (sqmul sqe1 sqe2) (sqmul sqe2 sqe1)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Dyer-Lashof operations
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (dl-adem2 a b)
  (define (iter stk i)
    (if (> i (- a b 1))
        (reverse stk)
        (if (zero? (binom2 (- i b 1) (- (* i 2) a)))
            (iter stk (+ i 1))
            (iter (cons (list (- (+ a b) i) i) stk)
                  (+ i 1)))))
  (cond ((< a b) '())  ; zero
        ((<= a (* b 2))
         (list a b))
        (else
         (iter '() (quotient (+ a 1) 2)))))

(define (dl-find-place dlmon)
  (define (search-iter head curr tail)
    (cond ((null? tail) (list dlmon '(()) '()))
          ((> curr (* 2 (car tail)))
           (list (reverse head)
                 (list curr (car tail))
                 (cdr tail)))
          (else
           (search-iter (cons curr head)
                        (car tail)
                        (cdr  tail)))))
  (if (null? dlmon)
      (list '() '(()) '())
```

```
      (search-iter '() (car dlmon) (cdr dlmon))))

(define (dl-apply-adem parts)
  (if (and (pair? (cadr parts))
           (not (null? (caadr parts))))
      (list (car parts)
            (dl-adem2 (caadr parts) (cadadr parts))
            (caddr parts))
      parts))

(define (dl-adem-monom dlmonom)
  (define (iter res rem)
    (if (null? rem)
        (remove-same-pairs (reverse res))
        (let ((parts (dl-find-place (car rem))))
          (if (null? (caadr parts))
              (iter (cons (car parts) res)
                    (cdr rem))
              (let ((exp (dl-apply-adem parts)))
                (iter res
                      (append
                       (develop-second
                        (dl-apply-adem parts))
                       (cdr rem)))))))))
  (iter '() (list dlmonom)))

(define (dlleq a b)
  (let ((la (length a)) (lb (length b)))
    (cond ((null? a) #t)
          ((null? b) #f)
          ((< la lb) #t)
          ((> la lb) #f)
          ((> (car a) (car b)) #t)
          ((< (car a) (car b)) #f)
          (else (dlleq (cdr a) (cdr b))))))
(define dl
  (lambda dlmonom
    (sqsort (dl-adem-monom dlmonom))))

(define (dlsort monoms)
  (sort monoms dlleq))

(define dladd
  (lambda exprs
```

```
    (dlsort (remove-same-pairs (apply append exprs)))))

(define (dl-normalize dlmlist)
  (define (iter res rem)
    (if (null? rem)
        (remove-same-pairs res)
        (iter (append
               (apply dl (car rem))
               res)
              (cdr rem))))
  (iter '() dlmlist))

;;
;; Action of the Steenrod algebra on Dyer-Lashof operations
;;  ( Nishida relations )
;;
(define (sqdl2 sqi dlj)
  ;; sqi, dlj are integers
  ;; (sqdl2 i k) means Sq^i Q^k
  ;; result is ((a_1 b_1) ... (a_n b_n))
  ;; meaning sum of Q^{a_1} Sq^{b_1}, ... , Q^{a_n} Sq^{b_n}
  (define (iter res k)
    (if (> (* k 2) sqi)
        (reverse res)
        (if (zero? (binom2 (- dlj sqi) (- sqi (* 2 k))))
            (iter res (+ k 1))
            (iter (cons (list (+ (- dlj sqi) k) k) res)
                  (+ k 1)))))
  (if (> sqi dlj)
      '()
      (iter '() 0)))

(define (sqdl-sub1 sqirevdlm dli)
  ;; dlsqi is
  ;; (sqi dli-n ... dli-1)
  ;; input: (i j_n ... j_1), k
  ;; meaning :  Q^{j_1}...Q^{j_n} Sq^i * Q^k
  ;; output: list of (b a j_n ... j_1)
  ;; meaning:  \sum (Q^{j_1}...Q^{j_n} Q^a Sq^b)
  (let ((x (sqdl2 (car sqirevdlm) dli)))
    (if (null? x)
        '()   ; zero
        (map (lambda (y) (pop-push y (cdr sqirevdlm))) x))))
```

```
(define (sqdl-sub2 sqirevdlmlist dli)
  ;; input : list of (i j_n ... j_1), k
  ;; meaning: \sum( Q^{j_1}...Q^{j_n} Sq^i ) ,  Q^k
  ;; output: list of (b a j_n ... j_1)
  ;; meaning: \sum (Q^{j_1}...Q^{j_n} Q^a Sq^b)
  (define (iter res rem)
    (if (null? rem)
        res
        (let ((newresult (sqdl-sub1 (car rem) dli)))
          (iter (append newresult res)
                (cdr rem)))))
  (iter '() sqirevdlmlist))

(define (sqdl-sub3 sqirevdlmlist dlm)
  ;; input : list of (i j_n ... j_1) , (k_1 ... k_m)
  ;; meaning: \sum(Q^{j_1}...Q^{j_n} Sq^i),  Q^{k_1}...Q^{k_m}
  ;; output: list of (b a_m ... a_1 j_n ... j_1)
  ;; meaning: \sum (Q^{j_1}...Q^{j_n}Q^{a_1}...Q^{a_m} Sq^b)
  (if (null? dlm)
      sqirevdlmlist
      (let ((x (sqdl-sub2 sqirevdlmlist (car dlm))))
        (sqdl-sub3 x (cdr dlm)))))

(define (sqidlm sqi dlm)
  ;; input :  i, (j_1 ... j_m)
  ;; meaning: Sq^i Q^{j_1}...Q^{j_m}
  ;; output: list of (j k_m ... k_1)
  ;; meaning: \sum (Q^{k_1}...Q^{k_m} Sq^j)
  (let ((x (sqdl-sub3 (list (list sqi)) dlm)))
    (map (lambda (y) (list (reverse (cdr y))
                           (car y)))
         x)))

(define (sqi-dlm-sqm sqi dlm sqm)
  ;; input : i , (j_1 ... j_m), (k_1 ... k_r)
  ;; meaning: Sq^i  Q^{j_1}...Q^{j_m}  Sq^{k_1}...Sq^{k_r}
  ;; output: list of ((l_m ... l_1)(k k_1 ... k_r))
  ;; meaning: \sum (Q^{l_1}...Q^{l_m} Sq^k Sq^{k_1} ... Sq^{k_r})
  (let ((x (sqidlm sqi dlm)))
    (map (lambda (y)
           (list (cdr y)
                 (if (zero? (car y))
                     sqm
                     (cons (car y) sqm))))
```

```
        x)))

(define (sqi-dlsqe sqi dlsqmlist)
  ;; input : i, list of ( (j_1 ... j_m) (k_1 ... k_r) )
  ;; meaning: Sq^i \sum( Q^{j_1}...Q^{j_1} Sq^{k_1}...Sq^{k_r} )
  ;; output:
  (let ((x (map (lambda (y)
                  (sqi-dlm-sqm sqi (car y) (cadr y)))
                dlsqmlist)))
    (display x)
    (remove-same-pairs x)))

(define (sqidlm sqi dlm)
  ;; input : i, (j_1 ... j_m)
  ;; meaning: Sq^i Q^{j_1} ... Q^{j_m}
  ;; output: list of ((k_1 ... k_m) b)
  ;; meaning: \sum (Q^{k_1} ... Q^{k_m} Sq^b)
  (let ((x (sqdl-sub3 (list (list sqi)) dlm)))
    (map (lambda (y) (list (reverse (cdr y))
                           (car y)))
         x)))

(define sqi 3)
(define dlsqmlist '(((20 18) (8 3))))

(define (take-zero-dim dlmsqilist)
  (define (iter res rem)
    (if (null? rem)
        (reverse res)
        (if (zero? (cadar rem))
            (iter (cons (caar rem) res)
                  (cdr rem))
            (iter res (cdr rem)))))
  (iter '() dlmsqilist))

(define (sqidle sqi dle)
  (define (iter res rem)
    (if (null? rem)
        (remove-same-pairs res)
        (let ((newresult
               (sqidlm sqi (car rem))))
          (iter (pop-push
                 (take-zero-dim newresult)
                 res)
```

```
                (cdr rem)))))
  (iter '() dle))

(define (sqmdle sqm dle)
  (if (null? sqm)
      (remove-same-pairs dle)
      (sqmdle (cdr sqm)
              (sqidle (car sqm)
                      dle))))

(define (sqmdlm sqm dlm)
  (sqmdle sqm (list dlm)))

;;
;; sq generators
;;
(define (sqm-excess sqm)
  (if (> (length sqm) 1)
      (apply - sqm)
      (car sqm)))

(define (sqm-dim sqm)
  (apply + sqm))

;;(define *sq-table-save* (make-vector (+ **max-dim** 1) #t))
;;(if (defined? '*sq-table*)
;;    (set! *sq-table-save* *sq-table*))
;;(define *sq-table* (make-vector (+ **max-dim** 1) #t))
;;(set! *sq-table* *sq-table-save*)
;;(define *sq-table-save* #f)

(define sq-leq sqleq)

(define (make-e-seq-once etop deg len res)
  (let ((weight (- (expt 2 len) 1)))
    (if (> (* weight etop) deg)
        res
        (let ((lower (make-e-seq-len 0
                                     (- deg (* weight etop))
                                     (- len 1))))
          (make-e-seq-once
            (+ etop 1)
            deg
            len
```

```
            (append (map (lambda (x) (cons etop x)) lower)
                    res))))))

(define (make-e-seq-len etop deg len)
  (cond ((= len 1) (list (list deg)))
        (else
         (make-e-seq-once etop deg len  '()))))

(define (make-e-seq deg)
  (define (log2 n)
    (if (< n 2)
        0
        (+ (log2 (quotient n 2)) 1)))
  (define (iter res len)
    (if (zero? len)
        res
        (let ((newres (make-e-seq-len 1 deg len)))
          (iter (append newres res) (- len 1)))))
  (iter '() (log2 (+ deg 1))))


(define (expn-to-expr expn)
  (define (iter res rem iold)
    (if (null? rem)
        res
        (let ((inew (+ iold iold (car rem))))
          (iter (cons inew res) (cdr rem) inew))))
  (iter '() expn 0))

(define (make-gen-sq deg)
  (sqsort (map expn-to-expr (make-e-seq deg))))

(define (make-sq-table-at deg)
  (if (pair? (vector-ref *sq-table* deg))
      (vector-ref *sq-table* deg)
      (begin
        (vector-set! *sq-table*
                     deg
                     (make-gen-sq deg)))))

(define (make-sq-table stopdeg)
  (define (iter k)
    (if (> k stopdeg)
        (vector-ref *sq-table* stopdeg)
```

```
        (begin
          (make-sq-table-at k)
          (iter (+ k 1)))))
  (iter 1))

(define sq-table make-sq-table)

;;
;; dl generators (type (i,j) only)
;;
;(define *dl-table* (make-vector 100))

(define (dl-excess dlm)
  (if (= (length dlm) 1)
      (car dlm)
      (apply - dlm)))

(define (dl2-table dim)
  (define (iter res k)
    (if (> (- dim k) (* k 2))
        res
        (iter (cons (list (- dim k) k) res)
              (- k 1))))
  (let ((i (quotient dim 2)))
    (iter '() i)))

;;
;;  sqm - dlm test in a dimension
;;

(define (sqm-dlmlist-test sqm dlmlist)
  (define (iter rem)
    (if (null? rem)
        #t
        (begin
          (display "  Q : ")
          (display (car rem))
          (display " => ")
          (display
            (if *debug*
                (dl-normalize (sqmdlm sqm (car rem)))
                (filter-kervaire
                 (dl-normalize (sqmdlm sqm (car rem))))))
          (newline)
```

```
          (iter (cdr rem)))))
  (display "Sq ")
  (display sqm)
  (display " maps")
  (newline)
  (iter dlmlist))


(define (sqm-dlm-test sqdim dldim)
  (let ((sqm-list (sq-table sqdim))
        (dlm-list (dl2-table dldim)))
    (map (lambda (sqm) (sqm-dlmlist-test sqm dlm-list))
         sqm-list)))

(define (sqm-dlm-test-aux sqmlist dlmlist)
  (map (lambda (sqm) (sqm-dlmlist-test
                      sqm
                      dlmlist))
       sqmlist))

;;
;; make cohomology classes including kervaire classes
;;

(define (make-kervaire-cohomology-kdim dim kervdim)
  (define (iter res rem)
    (if (null? rem)
        (reverse res)
        (if (> (sqm-excess (car rem)) kervdim)
            (iter res (cdr rem))
            (iter (cons (car rem) res)
                  (cdr rem)))))
  (iter '() (sq-table (- dim kervdim))))

(define (make-kervaire-cohomology dim)
  (define (iter res kdim)
    (if (>= kdim dim)
        res
        (let ((new-result
               (make-kervaire-cohomology-kdim dim kdim)))
          (iter (append res new-result)
                (+ (* kdim 2) 2)))))
  (iter '() 2))
;;
```

```
(define (sqdl-test dim)
  (let ((sqmlist (make-kervaire-cohomology dim))
        (dlmlist (dl2-table dim)))
    (sqm-dlm-test-aux sqmlist dlmlist)))

(define (filter-kervaire dlmlist)
  (define (iter res rem)
    (if (null? rem)
        (reverse res)
        (if (= (caar rem) (cadar rem))
            (iter (cons (car rem) res) (cdr rem))
            (iter res (cdr rem)))))
  (iter '() dlmlist))

;;(define (st dim)
;;  (filter-kervaire (sqdl-test dim)))
(define st sqdl-test)

;;
;;
;;
(define (minimum-sqm right leftdim)
  (if (>= leftdim (* 6 (car right)))
      (let ((x (if (null? right)
                   1
                   (* 2 (car right)))))
        (minimum-sqm (cons x right)
                     (- leftdim x)))
      (if (>= leftdim (* 2 (car right)))
          (cons leftdim right)
          #f)))

(define (next-sqm sqm)
  (define (right-iter leftdim right)
    (if (null? right)
        (list leftdim)
        (if (>= leftdim (+ 3 (* 2 (car right))))
            (minimum-sqm
             (cons (+ 1 (car right))
                   (cdr right))
             (- leftdim 1))
            (right-iter (+ leftdim (car right)) (cdr right)))))
  (if (null? (cdr sqm))
      #f
```

```
      (right-iter 0 sqm)))

;;
;;  ideal test
;;
(define (sqm-ideal sqm plusdeg)
  (define (iter rem)
    (if (null? rem)
        #t
        (begin
          (display "  ")
          (display (car rem))
          (display " : ")
          (display (sqmul (list sqm) (list (car rem))))
          (newline)
          (iter (cdr rem)))))

  (iter (sq-table plusdeg)))
;;
;;
;; ksqm : (6 (4 2))
(define (ksqm-leq? ksqm1 ksqm2)
  (cond ((> (car ksqm1) (car ksqm2)) #t)
        ((< (car ksqm1) (car ksqm2)) #f)
        (else (sqleq (cadr ksqm1) (cadr ksqm2)))))

(define (ksqm-geq? ksqm1 ksqm2)
  (cond ((> (car ksqm2) (car ksqm1)) #t)
        ((< (car ksqm2) (car ksqm1)) #f)
        (else (sqgeq (cadr ksqm1) (cadr ksqm2)))))



;; ksqe : list of ksqm
(define (normalize-ksqe ksqe)
  (sort ksqe ksqm-leq?))
(define (add-ksqe ksqe1 ksqe2)
  (define (iter rem1 rem2 stk)
    (cond ((null? rem1) (pop-push stk rem2))
          ((null? rem2) (pop-push stk rem1))
          ((equal? (car rem1) (car rem2))
           (iter (cdr rem1) (cdr rem2) stk))
          ((ksqm-leq? (car rem1) (car rem2))
           (iter (cdr rem1) rem2 (cons (car rem1) stk)))
```

```
          ((ksqm-leq? (car rem2) (car rem1))
           (iter rem1 (cdr rem2) (cons (car rem2) stk)))
          (else
           (iter (cdr rem1) (cdr rem2) stk))))
  (iter ksqe1 ksqe2 '()))


(define (make-kervaire-cohomology-with-kdim dim)
  (define (iter res kdim)
    (if (>= kdim dim)
        res
        (let ((newres (make-kervaire-cohomology-kdim dim kdim)))
          (iter (append res
                        (map (lambda (x) (list kdim x)) newres))
                (+ (* kdim 2) 2)))))
  (iter '() 2))

(define (sqsort sqmonoms)
  (sort sqmonoms sqm>?))

;;
;; Sweeping
;;

;;(require-library "compat.ss")
;;(require-library "macro.ss")
;;
;;(define (lds) (load "sweep.scm"))
;;

;;
(define (kerv-sqm-leq? ksqm1 ksqm2)
  (cond ((< (car ksqm1) (car ksqm2)) #t)
        ((> (car ksqm1) (car ksqm2)) #f)
        ((equal? (cadr ksqm1) (cadr ksqm2)) #t)
        (else (sqleq (cadr ksqm1) (cadr ksqm2)))))

(define (kerv-sqm-geq? ksqm1 ksqm2)
  (cond ((< (car ksqm2) (car ksqm1)) #t)
        ((> (car ksqm2) (car ksqm1)) #f)
        ((equal? (cadr ksqm1) (cadr ksqm2)) #t)
        (else (sqgeq (cadr ksqm1) (cadr ksqm2)))))

(define (kerv-sqe-leq? ksqe1 ksqe2)
```

```
  (cond ((eq? ksqe1 ksqe2) #t)
        ((null? ksqe1) #t)
        ((null? ksqe2) #f)
        ((equal? (car ksqe1) (car ksqe2))
         (kerv-sqe-leq? (cdr ksqe1) (cdr ksqe2)))
        ((kerv-sqm-leq? (car ksqe1) (car ksqe2)) #t)
        (else #f)))

(define (kerv-sqe-geq? ksqe1 ksqe2)
  (cond ((eq? ksqe2 ksqe1) #t)
        ((null? ksqe2) #t)
        ((null? ksqe1) #f)
        ((equal? (car ksqe1) (car ksqe2))
         (kerv-sqe-geq? (cdr ksqe1) (cdr ksqe2)))
        ((kerv-sqm-geq? (car ksqe1) (car ksqe2)) #t)
        (else #f)))

(define (kerv-sqe-add ksqe1 ksqe2)
  (define (iter rem1 rem2 stk)
    (cond ((null? rem1) (pop-push stk rem2))
          ((null? rem2) (pop-push stk rem1))
          ((equal? (car rem1) (car rem2))
           (iter (cdr rem1) (cdr rem2) stk))
          ((kerv-sqm-leq? (car rem1) (car rem2))
           (iter (cdr rem1) rem2 (cons (car rem1) stk)))
          (else (iter rem1 (cdr rem2) (cons (car rem2) stk)))))
  (iter ksqe1 ksqe2 '()))

(define relation-leq? kerv-sqe-leq?)
(define relation-geq? kerv-sqe-geq?)
;;
;; raw-data
;;
(define (build-raw-data-list dim)
  (let ((sqmlist (make-kervaire-cohomology dim)) ;; Sq^I Kerv
        (dlmlist (dl2-table dim)))               ;; DL op of len=2
    (define (iter result rem)
      (if (null? rem)
          (reverse result)
          (let* ((new (map (lambda (x)
                             (sqmdle (car rem) (list x)))
                           dlmlist))
                 (fun (list (- dim (apply + (car rem)))
                            (car rem))))
```

```
            (iter (cons (list fun new)
                        result)
                  (cdr rem)))))
    (iter '() sqmlist)))

(define (dle->number dle)
  (define (iter res rem)
    (if (null? rem)
        res
        (cond ((null? (car rem)) (iter res (cdr rem)))
              ((and (>= (caar rem) (cadar rem))
                    (2power-minus-one? (cadar rem)))
               (iter (- 1 res) (cdr rem))) ;; toggle 0 <-> 1 in res
              (else (iter res (cdr rem))))))
  (iter 0 dle))

(define (build-raw-data dim)
  (let ((data-lst (build-raw-data-list dim)))
    (map (lambda (x)
           (list (list (car x))
                 (list->vector
                  (map dle->number (cadr x)))))
         data-lst)))

(define (add-vec-mod2 vec1 vec2)
  (let* ((len (vector-length vec1))
         (v (make-vector len)))
    (do ((i 0 (+ i 1)))
        ((= i len) v)
      (vector-set! v
                   i
                   (remainder (+ (vector-ref vec1 i)
                                 (vector-ref vec2 i))
                              2)))))

(define (add-row-data row-data1 row-data2)
  (let ((ksq (kerv-sqe-add (car row-data1)
                           (car row-data2)))
        (v (add-vec-mod2 (cadr row-data1)
                         (cadr row-data2))))
    (list ksq v)))

;;
;; Sweep to obtain zero functions
```

```
;;

;; search nonzero element vertically
;; starting from start-row at column j
(define (search-nonzero-vertical tab-data start-row j)
  ;; start-row is pointer
  (define (iter cur-row)
    (if (not (zero? (vector-ref (cadar cur-row) j)))
        cur-row
        (if (null? (cdr cur-row))
            #f
            (iter (cdr cur-row)))))
  (iter start-row))

(define (sweep-by-one-row tab-data pivot-row j)
  (define (iter stk rem)
    (cond ((null? rem) (reverse stk))
          ((eq? rem pivot-row)   ;; pivot is not necessary
           (iter stk (cdr rem))) ;; for zero functions
          ((zero? (vector-ref (cadar rem) j))
           (iter (cons (car rem) stk) (cdr rem)))
          (else
           (let ((new-row-data
                  (add-row-data (car pivot-row)
                                (car rem))))
             (iter (cons new-row-data stk)
                   (cdr rem))))))
  (iter '() tab-data))

(define (sweep-column tab-data col)
  (let ((row (search-nonzero-vertical tab-data tab-data col)))
    (if row
        (sweep-by-one-row tab-data row col)
        tab-data)))  ;; when all vertical elements are 0

(define (sweep-out tab-data)
  (define (iter i vlen res)
    (if (or (= i vlen)
            (null? res))
        res
        (iter (+ i 1)
              vlen
              (sweep-column res i))))
  (if (null? tab-data)
```

```
      '()
      (let ((veclen (vector-length (cadar tab-data))))
        (iter 0 veclen tab-data))))

(define (build-zero-functions dim)
  (if (not (vector-ref *zero-functions* dim))
      (vector-set! *zero-functions*
                   dim
                   (sort (map (lambda (x)
                                (sort x kerv-sqm-leq?))
                              (map car
                                   (sweep-out
                                    (build-raw-data dim))))
                         kerv-sqe-leq?)))
  (vector-ref *zero-functions* dim))

(define bzf build-zero-functions)

;;
;;  Build database from known relations in lower dimensions
;;
(define (over-excess? ksqm)
  (if (> (length (cadr ksqm)) 1)
      (> (apply - (cadr ksqm)) (car ksqm))
      (> (caadr ksqm) (car ksqm))))

(define (derive-relation-by-sqm orig-rel sqm)
  (define (iter rem res)
    (if (null? rem)
        res
        (let ((new-sqe
               (sort (sqmulmonos sqm (cadar rem)) sqleq)))
          (iter (cdr rem)
                (kerv-sqe-add
                 res
                 (map (lambda (x)
                        (list (caar rem) x))
                      new-sqe))))))
  (remove-if over-excess?
             (iter orig-rel '())))

(define (derive-relation-list-by-sqm orig-rel-list sqm)
  (define (iter res rem)
    (if (null? rem)
```

```
        (sweep-relations
          (reverse res))
        (let ((rel (derive-relation-by-sqm (car rem) sqm)))
          (if (null? rel)
              (iter res (cdr rem))
              (iter (cons rel res) (cdr rem))))))
  (remove-if null?
             (iter '() orig-rel-list)))

;;1 the most time consuming function
(define (derived-relations dim)
  (define (iter res k)
    (if (< (- dim k) 9)
        res
        (let ((rels (derive-relation-list-by-sqm
                     (build-zero-functions (- dim k))
                     (list k)) ))
          (iter (append rels res) (+ k k)))))
  (if (not (vector-ref *derived-zero-functions* dim))
      (let ((dzfs (iter '() 1)))
        (vector-set! *derived-zero-functions*
                     dim
                     (sweep-relations dzfs))))
  (vector-ref *derived-zero-functions* dim))

(define (generating-relations dim)
  (if (not (vector-ref *generating-zero-functions* dim))
      (let ((genrels
             (sweep-relations-with-relation-db
               (derived-relations dim)
               (build-zero-functions dim))))
        (vector-set! *generating-zero-functions* dim genrels)))
  (vector-ref *generating-zero-functions* dim))

(define gr generating-relations)
;
;;
(define (unique-relations relation-list)
  (define (iter rem stk)
    (if (null? rem)
        (reverse stk)
        (if (equal? (car rem) (car stk))
            (iter (cdr rem) stk)
            (iter (cdr rem) (cons (car rem) stk)))))
```

```
  (if (null? relation-list)
      relation-list
      (iter (cdr relation-list)
            (list (car relation-list)))))

(define (sort-relations relation-list)
  (unique-relations
   (sort relation-list relation-leq?)))

(define (reverse-sort-relations relation-list)
  (unique-relations
   (sort relation-list relation-geq?)))

(define (derive-relation-by-dim-raw orig-rel dim-diff)
  (let ((sqmlist (sq-table dim-diff)))
    (remove-if
     null?
     (sort-relations
      (map (lambda (sqm)
             (derive-relation-by-sqm orig-rel sqm))
           sqmlist)))))

(define (derive-relation-list-raw rel-list sqm)
  (sort-relations
   (remove-if
     null?
     (map (lambda (rel)
            (derive-relation-by-sqm rel sqm))
          rel-list))))

(define (derive-relation-from-below rels-vec dim)
  (define (derive-iter res k)
    (if (< (- dim k) 9)
        res
        (let ((newrels
               (derive-relation-list-raw
                  (vector-ref rels-vec (- dim k))
                  (list k)) ))
          (derive-iter (append newrels res) (* 2 k)))))
  (let ((rels-raw (derive-iter '() 1)))
    (if (null? rels-raw)
        rels-raw
        (sweep-relations
         (derive-iter '() 1)))))
```

```
(define (derive-raw-from-rel-list rel-list target-dim)
  (define (iter rem res)
    (if (null? rem)
        res
        (let ((dim-diff (- target-dim
                           (+ (caaar rem)
                              (apply + (cadaar rem))))))
          (cond ((zero? dim-diff)
                 (iter (cdr rem)
                       (cons (car rem) res)))
                ((> dim-diff 0)
                 (iter (cdr rem)
                       (append
                        (derive-relation-by-dim-raw (car rem)
                                                    dim-diff)
                        res)))
                (else (error)))))))
  (sort-relations
   (remove-if null? (iter rel-list '()))))


;;
(define (sweep-relation-by-pivot pivot-rel relation-list)
  (define (iter rem stk)
    (if (null? rem)
        (remove-if null?
                   (reverse stk))
        (if (sorted-member (car pivot-rel)
                           (car rem)
                           kerv-sqm-leq?)
            (let ((new-rel (kerv-sqe-add pivot-rel
                                         (car rem))))
              (if (null? new-rel)
                  (iter (cdr rem) stk)
                  (iter (cdr rem)
                        (cons new-rel stk))))
            (iter (cdr rem)
                  (cons (car rem) stk)))))
  (sort-relations
   (iter relation-list '())))
;;
(define (sweep-relations-with-relation-db db-list rel-list)
```

```
  (define (car-assoc obj lst)
    (if (null? lst)
        #f
        (if (equal? obj (caar lst))
            lst
            (car-assoc obj (cdr lst)))))
  (define (iter-forward rem stk)
    (if (null? rem)
        stk
        (if (null? (car rem))
            (iter-forward (cdr rem) stk)
            (let ((rel (car-assoc (caar rem) db-list)))
              (if rel
                  (iter-forward (cons
                                 (kerv-sqe-add (car rel) (car rem))
                                 (cdr rem))
                                stk)
                  (iter-forward (cdr rem)
                                (cons (car rem) stk)))))))
  (sweep-relations
    (iter-forward rel-list '())))



;;
(define (sweep-relations-forward relation-list)
  (define (iter rem stk)
    (if (or (null? rem) (null? (cdr rem)))
        (pop-push stk rem)
        (let ((one-step-result
               (remove-if null?
                          (sort-relations
                           (sweep-relation-by-pivot (car rem)
                                                    (cdr rem))) ) ))
          (iter one-step-result (cons (car rem) stk)))))
  (iter relation-list '()))

(define (sweep-relations-backward relation-list)
  (define (iter rem stk)
    (if (null? (cdr rem))
        (pop-push stk rem)
        (let ((one-step-result
               (remove-if null?
```

```
                        (reverse
                         (sort-relations
                          (sweep-relation-by-pivot (car rem)
                                                   (cdr rem)))))))
          (iter one-step-result (cons (car rem) stk)))))
  (reverse
   (iter (reverse relation-list) '())))

(define (sweep-relations relations)
  (if (null? relations)
      relations
      (sort-relations
       (sweep-relations-backward
        (sweep-relations-forward relations)))))
;;
(define (derive-relation-by-dim orig-rel diff-dim)
  (sort-relations
   (sweep-relations
    (derive-relation-by-dim-raw orig-rel diff-dim))))

(define (derive-relation-db orig-rellist target-dim)
  (sort-relations
   (sweep-relations
    (derive-raw-from-rel-list orig-rellist target-dim))))

(define drdbr derive-relation-by-dim-raw)
(define drdb derive-relation-db)
;;
;;
;; Determine if new relations can be deduced
;; by known relations
;;
(define (simplify-new-relation rel-db new-rel)
  (define (check-db-rel rem res)
    (if (or (null? rem) (null? res))
        res
        (let ((head (member (caar rem) res)))
          (if head
              (check-db-rel
               (cdr rem)
               (kerv-sqe-add res (car rem)))
              (check-db-rel
               (cdr rem)
               res))))))
```

```
 (check-db-rel rel-db new-rel))

(define (simplify-new-relation-list rel-db new-rel-list)
 (define (iter rem stk)
   (if (null? rem)
       (reverse stk)
       (let ((one-result
              (simplify-new-relation rel-db (car rem))))
         (if (null? one-result)
             (iter (cdr rem) stk)
             (iter (cdr rem) (cons one-result stk))))))
 (iter new-rel-list '()))

(define snrl simplify-new-relation-list)
(define (tst rdb dim)
 (let ((new-rels (bzf dim)))
   (if (null? new-rels)
       new-rels
       (snrl (drdb rdb dim) new-rels))))

(define (proceed dim)
 (let ((rel-db
        (derive-relation-from-below *relsvec* dim))
       (new-rels (bzf dim)))
   (if (null? new-rels)
       (vector-set! *relsvec* dim rel-db)
       (let ((simplified-rels
              (simplify-new-relation-list rel-db new-rels)))
         (vector-set! *relsvec*
                      dim
                      (sort-relations
                       (append simplified-rels
                               rel-db)))
         (vector-ref *relsvec* dim)))))

;;
;; database of relations
;;
(define *relsvec* (make-vector (+ **max-dim** 1) #f))
(define r-9 '(((6 (2 1))(2 (4 2 1)))))
(vector-set! *relsvec* 9 r-9)
(define r-16 '(((6 (8 2)) (6 (7 3)) (2 (8 4 2)))))
(define r-17 '(((14 (2 1)) (2 (8 4 2 1)))))
(define r-19 '(((14 (4 1)) (6 (8 4 1)))))
```

```
(define r-20 '(((14 (4 2)) (6 (8 4 2)))))
;;!!
(define r-32 '(((14 (16 2)) (14 (15 3))
                            (6 (16 7 3)) (2 (16 8 4 2)))))
(define r-33 '(((30 (2 1)) (2 (16 8 4 2 1)))))
;;!!
(define r-34 '(((14 (16 4))
                (14 (15 5)) (14 (14 6)) (6 (16 8 4)))))
(define r-35  '(((30 (4 1)) (6 (16 8 4 1)))))
(define r-36 '(((30 (4 2)) (6 (16 8 4 2)))))
(define r-39 '(((30 (8 1))(14 (16 8 1)))))
(define r-40 '(((30 (8 2)) (14 (16 8 2)))))
(define r-42 '(((30 (8 4)) (14 (16 8 4)))))
;;!!
(define r-64 '(((30 (32 2)) (30 (31 3))
                            (14 (30 15 5)) (6 (32 16 7 3))
                            (2 (32 16 8 4 2)))))
(define r-65 '(((62 (2 1)) (2 (32 16 8 4 2 1)))))
;;!!
(define r-66 '(((30 (32 4)) (30 (31 5)) (30 (30 6))
                            (14 (32 15 5)) (14 (32 14 6))
                            (6 (32 16 8 4)))))
(define r-67 '(((62 (4 1)) (6 (32 16 8 4 1)))))
(define r-68 '(((62 (4 2)) (6 (32 16 8 4 2)))))
;;!!
(define r-70 '(((30 (32 8))
                (30 (31 9)) (30 (30 10))
                (30 (28 12)) (14 (32 16 8)))))
(define r-71 '(((62 (8 1)) (14 (32 16 8 1)))))
(define r-72 '(((62 (8 2)) (14 (32 16 8 2)))))
(define r-74 '(((62 (8 4)) (14 (32 16 8 4)))))
(define r-79 '(((62 (16 1)) (30 (32 16 1)))))
(define r-80 '(((62 (16 2)) (30 (32 16 2)))))
(define r-82 '(((62 (16 4)) (30 (32 16 4)))))
(define r-86 '(((62 (16 8)) (30 (32 16 8)))))

(define rdb9 r-9)
(define rdb16 (append r-16 rdb9))
(define rdb17 (append r-17 rdb16))
(define rdb19 (append r-19 rdb17))
(define rdb20 (append r-20 rdb19))
(define rdb32 (append r-32 rdb20))
(define rdb33 (append r-33 rdb32))
(define rdb34 (append r-34 rdb33))
```

```
(define rdb35 (append r-35 rdb34))
(define rdb36 (append r-36 rdb35))
(define rdb39 (append r-39 rdb36))
(define rdb40 (append r-40 rdb39))
(define rdb42 (append r-42 rdb40))
(define rdb64 (append r-64 rdb42))
(define rdb65 (append r-65 rdb64))
(define rdb66 (append r-66 rdb65))
(define rdb67 (append r-67 rdb66))
(define rdb68 (append r-68 rdb67))
(define rdb70 (append r-70 rdb68))
(define rdb71 (append r-71 rdb70))
(define rdb72 (append r-72 rdb71))
(define rdb74 (append r-74 rdb72))
(define rdb79 (append r-79 rdb74))
(define rdb80 (append r-80 rdb79))
(define rdb82 (append r-82 rdb80))
(define rdb86 (append r-86 rdb82))
;;
;; database valid upto dim 111
;;
(define rdb rdb86)

(define (prt-sqm sqm)
  (display "Sq")
  (display sqm))

(define (prt-rel rel)
  (define (dispkerv kdim)
    (if (> kdim 0)
        (begin
          (display ")K")
          (display kdim))))
  (define (iter rem prev-kdim)
    (if (null? rem)
        (dispkerv prev-kdim)
        (let ((kdim (caar rem)))
          (if (= kdim prev-kdim)
              (begin
                (display "+")
                (prt-sqm (cadar rem))
                (iter (cdr rem) kdim))
              (begin
                (dispkerv prev-kdim)
```

```
                (if (> prev-kdim 0)
                    (display " + "))
                (display "(")
                (prt-sqm (cadar rem))
                (iter (cdr rem) kdim))))))
  (if (null? rel)
      (display "0")
      (iter rel 0))
  (newline))

(define (prt-rel-list rel-list)
  (define (iter rem)
    (if (not (null? rem))
        (begin
          (prt-rel (car rem))
          (iter (cdr rem)))))
  (iter rel-list))

(define (extend-rdb orig-rdb start-dim end-dim)
  (define (iter dim rdb)
    (if (> dim end-dim)
        rdb
        (let ((new-rels (tst rdb dim)))
          (display dim)
          (if (bound? '*heapsize*)
              (begin
                (display " heap size=")
                (display *heapsize*)))
          (newline)
          ;;        (display new-rels)(newline)
          (prt-rel-list new-rels)
          (iter (+ dim 1) (append new-rels rdb)))))
  (iter start-dim orig-rdb))

(define (values ksqe)
  (let* ((dim (+ (caar ksqe)
                 (apply + (cadar ksqe))))
         (dlmlist (dl2-table dim)))
    (define (iter1 result rem-dle)
      (if (null? rem-dle)
          (reverse result)
          (let ((dlm-result
                 (iter2 (car rem-dle) ksqe '())))
            (iter1 (cons dlm-result result)
```

```
                (cdr rem-dle)))))
    (define (iter2 dlm ksqe-rem res)
      (if (null? ksqe-rem)
          res
          (let ((resksqm
                 (sqmdle (cadar ksqe-rem) (list dlm))))
            (iter2 dlm
                   (cdr ksqe-rem)
                   (sqadd resksqm res)))))
    (map dle->number (iter1 '() dlmlist))))

(define (sq-ideal sqe dim)
  (let ((dim-diff (- dim (apply + (car sqe)))))
    (map (lambda (y) (sqmul (list y) sqe))
         (sq-table dim-diff))))

(define (gen dim-low dim-high)
  (do ((i dim-low (+ i 1)))
      ((> i dim-high))
    (let ((x (generating-relations i)))
      (display "dim=")(display i)
      (if (null? x)
          (newline)
          (begin
            (display " ")
            (prt-rel-list x))))))
```