

```

-- file TypePack.Mesa
-- last modified by Satterthwaite, June 30, 1978 11:34 AM

DIRECTORY
  StringDefs: FROM "stringdefs",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  TypePackDefs: FROM "typepackdefs";

TypePack: PROGRAM IMPORTS StringDefs EXPORTS TypePackDefs =
BEGIN
  OPEN SymDefs, TypePackDefs;

  -- internal utilities

  HTHandle: TYPE = RECORD[
    stb: SymbolTableBase,
    hti: HTIndex];

  EqualIds: PROCEDURE [id1, id2: HTHandle] RETURNS [BOOLEAN] =
  BEGIN
    OPEN b1: id1.stb, b2: id2.stb;
    ss1, ss2: StringDefs.SubStringDescriptor;
    IF id1 = id2 THEN RETURN [TRUE];
    b1.SubStringForHash[@ss1, id1.hti]; b2.SubStringForHash[@ss2, id2.hti];
    RETURN [StringDefs.EqualSubStrings[@ss1, @ss2]]
  END;

  CTXHandle: TYPE = RECORD[
    stb: SymbolTableBase,
    ctx: CTXIndex];

  EqContexts: PROCEDURE [context1, context2: CTXHandle] RETURNS [BOOLEAN] =
  BEGIN
    OPEN b1: context1.stb, b2: context2.stb;
    ctx1, ctx2: CTXIndex;
    mdi1, mdi2: MDIndex;
    IF context1 = context2 THEN RETURN [TRUE];
    IF context1.stb = context2.stb THEN RETURN [FALSE];
    IF LOOPHOLE[context1.ctx, CARDINAL] <= 5*SIZE[simple CTXRecord]
      THEN RETURN [context1.ctx = context2.ctx]; -- predefined types
    WITH c1: (b1.ctxb+context1.ctx) SELECT FROM
      simple =>
      BEGIN mdi1 ← OwnMdi; ctx1 ← context1.ctx;
      END;
    included =>
      BEGIN mdi1 ← c1.ctxmodule; ctx1 ← c1.ctxmap;
      END;
    ENDCASE => ERROR;
    WITH c2: (b2.ctxb+context2.ctx) SELECT FROM
      simple =>
      BEGIN mdi2 ← OwnMdi; ctx2 ← context2.ctx;
      END;
    included =>
      BEGIN mdi2 ← c2.ctxmodule; ctx2 ← c2.ctxmap;
      END;
    ENDCASE => ERROR;
    RETURN [ctx1 = ctx2
      AND EqualIds[
        [context1.stb, (b1.mdb+mdi1).mdhti],
        [context2.stb, (b2.mdb+mdi2).mdhti]]
      AND
      ((b1.mdb+mdi1).mdStamp.zapped OR (b2.mdb+mdi2).mdStamp.zapped
      OR (b1.mdb+mdi1).mdStamp = (b2.mdb+mdi2).mdStamp)]
  END;

  -- type relations

  EquivalentTypes: PUBLIC PROCEDURE [type1, type2: TypeHandle] RETURNS [BOOLEAN] =
  BEGIN
    OPEN b1: type1.stb, b2: type2.stb;
    IF type1 = type2 OR type1.sei = typeANY OR type2.sei = typeANY
      THEN RETURN [TRUE];
    IF type1.sei = SEnull OR type2.sei = SEnull

```

```

THEN RETURN [type1.sei = type2.sei];
RETURN [WITH t1: (b1.seb+type1.sei) SELECT FROM
basic =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    basic => t1.code = t2.code,
    ENDCASE => FALSE,
enumerated =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    enumerated =>
      EqContexts[[type1.stb, t1.valuectx], [type2.stb, t2.valuectx]],
      ENDCASE => FALSE,
record =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    record =>
      EqContexts[[type1.stb, t1.fieldctx], [type2.stb, t2.fieldctx]],
      ENDCASE => FALSE,
pointer =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    pointer =>
      (t1.ordered = t2.ordered)
      AND EquivalentTypes[
        [type1.stb, b1.UnderType[t1.pointedTOTYPE]],
        [type2.stb, b2.UnderType[t2.pointedTOTYPE]]],
      ENDCASE => FALSE,
array =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    array =>
      t1.packed = t2.packed
      AND EquivalentTypes[
        [type1.stb, b1.UnderType[t1.componenttype]],
        [type2.stb, b2.UnderType[t2.componenttype]]]
      AND EquivalentTypes[
        [type1.stb, b1.UnderType[t1.indextype]],
        [type2.stb, b2.UnderType[t2.indextype]]],
      ENDCASE => FALSE,
arraydesc =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    arraydesc =>
      EquivalentTypes[
        [type1.stb, b1.UnderType[t1.describedType]],
        [type2.stb, b2.UnderType[t2.describedType]]],
      ENDCASE => FALSE,
transfer =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    transfer =>
      t1.mode = t2.mode
      AND EquivalentArgs[
        [type1.stb, t1.inrecord],
        [type2.stb, t2.inrecord]]
      AND EquivalentArgs[
        [type1.stb, t1.outrecord],
        [type2.stb, t2.outrecord]],
      ENDCASE => FALSE,
relative =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    relative =>
      EquivalentTypes[
        [type1.stb, b1.UnderType[t1.baseType]],
        [type2.stb, b2.UnderType[t2.baseType]]]
      AND EquivalentTypes[
        [type1.stb, b1.UnderType[t1.offsetType]],
        [type2.stb, b2.UnderType[t2.offsetType]]],
      ENDCASE => FALSE,
subrange =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM
    subrange =>
      EquivalentTypes[
        [type1.stb, b1.UnderType[t1.rangetype]],
        [type2.stb, b2.UnderType[t2.rangetype]]]
      AND
        (~t1.filled OR ~t2.filled
        OR (t1.origin = t2.origin AND t1.empty = t2.empty
        AND (t1.empty OR t1.range = t2.range))),
      ENDCASE => FALSE,
long =>
  WITH t2: (b2.seb+type2.sei) SELECT FROM

```

```

    long =>
      EquivalentTypes[
        [type1.stb, b1.UnderType[t1.rangetype]],
        [type2.stb, b2.UnderType[t2.rangetype]]],
      ENDCASE => FALSE,
    real =>
      WITH t2: (b2.seb+type2.sei) SELECT FROM
        real => TRUE,
      ENDCASE => FALSE,
    ENDCASE => FALSE]
  END;

```

```

ArgHandle: TYPE = RECORD[
  stb: SymbolTableBase,
  sei: recordCSEIndex];

```

```

EquivalentArgs: PROCEDURE [arg1, arg2: ArgHandle] RETURNS [BOOLEAN] =
  BEGIN
    OPEN b1: arg1.stb, b2: arg2.stb;
    sei1, sei2: ISEIndex;
    checkids: BOOLEAN;
    IF arg1.sei = SEnull OR arg2.sei = SEnull
      THEN RETURN [arg1.sei = arg2.sei];
    checkids ← ~(b1.seb+arg1.sei).unifield AND ~(b2.seb+arg2.sei).unifield;
    sei1 ← b1.FirstCtxSe[(b1.seb+arg1.sei).fieldctx];
    sei2 ← b2.FirstCtxSe[(b2.seb+arg2.sei).fieldctx];
    UNTIL sei1 = SEnull OR sei2 = SEnull
      DO
        IF ~EquivalentTypes[
            [arg1.stb, b1.UnderType[(b1.seb+sei1).idtype]],
            [arg2.stb, b2.UnderType[(b2.seb+sei2).idtype]]]
          OR (checkids
            AND (b1.seb+sei1).htptr # HTNull
            AND (b2.seb+sei2).htptr # HTNull
            AND ~EqualIds[
                [arg1.stb, (b1.seb+sei1).htptr],
                [arg2.stb, (b2.seb+sei2).htptr]]])
          THEN RETURN [FALSE];
        sei1 ← b1.NextSe[sei1]; sei2 ← b2.NextSe[sei2];
      ENDOLOOP;
    RETURN [sei1 = sei2]
  END;

```

```

AssignableTypes: PUBLIC PROCEDURE [typeL, typeR: TypeHandle] RETURNS [BOOLEAN] =
  BEGIN
    OPEN bL: typeL.stb, bR: typeR.stb;
    IF typeL = typeR OR typeL.sei = typeANY OR typeR.sei = typeANY
      THEN RETURN [TRUE];
    IF typeL.sei = SEnull OR typeR.sei = SEnull
      THEN RETURN [typeL.sei = typeR.sei];
    RETURN [WITH tL: (bL.seb+typeL.sei) SELECT FROM
      record =>
        WITH tR: (bR.seb+typeR.sei) SELECT FROM
          record =>
            EqContexts[[typeL.stb, tL.fieldctx], [typeR.stb, tR.fieldctx]]
          OR
            (WITH tR SELECT FROM
              linked => AssignableTypes[
                typeL,
                [typeR.stb, bR.UnderType[linktype]]],
            ENDCASE => FALSE),
          ENDCASE => FALSE,
      pointer =>
        WITH tR: (bR.seb+typeR.sei) SELECT FROM
          pointer =>
            (~tL.ordered OR tR.ordered)
            AND AssignableTypes[
              [typeL.stb, bL.UnderType[tL.pointedTOTYPE]],
              [typeR.stb, bR.UnderType[tR.pointedTOTYPE]]],
          ENDCASE => FALSE,
      arraydesc =>
        WITH tR: (bR.seb+typeR.sei) SELECT FROM
          arraydesc =>
            CommonTypes[

```

```

        [typeL.stb, bL.UnderType[tL.describedType]],
        [typeR.stb, bR.UnderType[tR.describedType]],
    ENDCASE => FALSE,
transfer =>
    WITH tR: (bR.seb+typeR.sei) SELECT FROM
        transfer =>
            (tL.mode = tR.mode OR (tL.mode = error AND tR.mode = signal))
            AND EquivalentArgs[
                [typeL.stb, tL.inrecord],
                [typeR.stb, tR.inrecord]]
            AND EquivalentArgs[
                [typeL.stb, tL.outrecord],
                [typeR.stb, tR.outrecord]],
    ENDCASE => FALSE,
relative =>
    WITH tR: (bR.seb+typeR.sei) SELECT FROM
        relative =>
            EquivalentTypes[
                [typeL.stb, bL.UnderType[tL.baseType]],
                [typeR.stb, bR.UnderType[tR.baseType]]]
            AND AssignableTypes[
                FullRangeType[[typeL.stb, bL.UnderType[tL.offsetType]]],
                FullRangeType[[typeR.stb, bR.UnderType[tR.offsetType]]]],
    ENDCASE => FALSE,
subrange => CoveringType[typeL, typeR],
long =>
    WITH tR: (bR.seb+typeR.sei) SELECT FROM
        long =>
            AssignableTypes[
                [typeL.stb, bL.UnderType[tL.rangetype]],
                [typeR.stb, bR.UnderType[tR.rangetype]]],
    ENDCASE => FALSE,
    ENDCASE => EquivalentTypes[typeL, typeR]
END;

```

```

CommonTypes: PROCEDURE [typeL, typeR: TypeHandle] RETURNS [BOOLEAN] =
    BEGIN
    OPEN bL: typeL.stb, bR: typeR.stb;
    IF typeL = typeR OR typeL.sei = typeANY OR typeR.sei = typeANY
    THEN RETURN [TRUE];
    RETURN [WITH tL: (bL.seb+typeL.sei) SELECT FROM
        array =>
            WITH tR: (bR.seb+typeR.sei) SELECT FROM
                array =>
                    tL.packed = tR.packed
                    AND EquivalentTypes[
                        [typeL.stb, bL.UnderType[tL.componenttype]],
                        [typeR.stb, bR.UnderType[tR.componenttype]]]
                    AND CoveringType[
                        [typeL.stb, bL.UnderType[tL.indextype]],
                        [typeR.stb, bR.UnderType[tR.indextype]]],
                ENDCASE => FALSE,
            ENDCASE => EquivalentTypes[typeL, typeR]
    END;

```

```

CoveringType: PROCEDURE [type1, type2: TypeHandle] RETURNS [BOOLEAN] =
    BEGIN
    OPEN b1: type1.stb, b2: type2.stb;
    RETURN [WITH t2: (b2.seb+type2.sei) SELECT FROM
        subrange =>
            WITH t1: (b1.seb+type1.sei) SELECT FROM
                subrange =>
                    IF ~t1.filled OR ~t2.filled OR t2.empty
                    OR (t1.origin = t2.origin AND t1.range >= t2.range)
                    THEN CoveringType[
                        [type1.stb, b1.UnderType[t1.rangetype]],
                        type2]
                    ELSE FALSE,
                ENDCASE =>
            CoveringType[
                type1,
                [type2.stb, b2.UnderType[t2.rangetype]]],
    ENDCASE => EquivalentTypes[type1, type2]
    END;

```

```
FullRangeType: PROCEDURE [type: TypeHandle] RETURNS [TypeHandle] =
  BEGIN
  OPEN b: type.stb;
  sei: CSEIndex;
  sei ← type.sei;
  DO
  WITH (b.seb+sei) SELECT FROM
    subrange => sei ← b.UnderType[rangetype];
  ENDCASE => EXIT;
  ENDOLOOP;
  RETURN [[type.stb, sei]]
  END;

END.
```