/\*\*Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Synopsis    [Derives a simple CNF for the AIG.]

  Description [The last argument lists the number of last outputs

  of the manager, which will not be converted into clauses.

  New variables will be introduced for these outputs.]

  SideEffects []

  SeeAlso     []

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Cnf\_Dat\_t \* Cnf\_DeriveSimple( Aig\_Man\_t \* p, int nOutputs )

{

    Aig\_Obj\_t \* pObj;

    Cnf\_Dat\_t \* pCnf;

    int OutVar, PoVar, pVars[32], \* pLits, \*\* pClas;

    int i, nLiterals, nClauses, Number;

    // count the number of literals and clauses

    nLiterals = 1 + 7 \* Aig\_ManNodeNum(p) + Aig\_ManCoNum( p ) + 3 \* nOutputs;

    nClauses = 1 + 3 \* Aig\_ManNodeNum(p) + Aig\_ManCoNum( p ) + nOutputs;

    // allocate CNF

    pCnf = ABC\_ALLOC( Cnf\_Dat\_t, 1 );

    memset( pCnf, 0, sizeof(Cnf\_Dat\_t) );

    pCnf->pMan = p;

    pCnf->nLiterals = nLiterals;

    pCnf->nClauses = nClauses;

    pCnf->pClauses = ABC\_ALLOC( int \*, nClauses + 1 );

    pCnf->pClauses[0] = ABC\_ALLOC( int, nLiterals );

    pCnf->pClauses[nClauses] = pCnf->pClauses[0] + nLiterals;

    // create room for variable numbers

    pCnf->pVarNums = ABC\_ALLOC( int, Aig\_ManObjNumMax(p) );

//    memset( pCnf->pVarNums, 0xff, sizeof(int) \* Aig\_ManObjNumMax(p) );

    for ( i = 0; i < Aig\_ManObjNumMax(p); i++ )

        pCnf->pVarNums[i] = -1;

    // assign variables to the last (nOutputs) POs

    Number = 1;

    if ( nOutputs )

    {

//        assert( nOutputs == Aig\_ManRegNum(p) );

//        Aig\_ManForEachLiSeq( p, pObj, i )

Number变量是一个计数器，通过遍历PO，Node，PI来给aig中的object标定其对应变量在CNF中的序号。PO对应的变量的序号是最靠前的。

//            pCnf->pVarNums[pObj->Id] = Number++;

        Aig\_ManForEachCo( p, pObj, i )

            pCnf->pVarNums[pObj->Id] = Number++;

    }

    // assign variables to the internal nodes

Node对应的变量的序号在PO后面

    Aig\_ManForEachNode( p, pObj, i )

        pCnf->pVarNums[pObj->Id] = Number++;

    // assign variables to the PIs and constant node

    Aig\_ManForEachCi( p, pObj, i )

CI/PI 对应的变量的序号在最后

        pCnf->pVarNums[pObj->Id] = Number++;

    pCnf->pVarNums[Aig\_ManConst1(p)->Id] = Number++;

CNF中的总变量数（Var）跟aig中的PO, node, PI/CI的总数是一样的。

    pCnf->nVars = Number;

/\*

    // print CNF numbers

    printf( "SAT numbers of each node:\n" );

    Aig\_ManForEachObj( p, pObj, i )

        printf( "%d=%d ", pObj->Id, pCnf->pVarNums[pObj->Id] );

    printf( "\n" );

\*/

    // assign the clauses

pClauses是一个二级指针，pLits是一个指针，指向某个Lit(一般为int)

    pLits = pCnf->pClauses[0];

    pClas = pCnf->pClauses;

    Aig\_ManForEachNode( p, pObj, i )

这里是通过Tseitin Encoding来实现aig到CNF的转换

    {

A yellow rectangular object with colorful text

Description automatically generated        OutVar   = pCnf->pVarNums[ pObj->Id ];

        pVars[0] = pCnf->pVarNums[ Aig\_ObjFanin0(pObj)->Id ];

        pVars[1] = pCnf->pVarNums[ Aig\_ObjFanin1(pObj)->Id ];

        // positive phase

        \*pClas++ = pLits;

        \*pLits++ = 2 \* OutVar;

A yellow rectangular object with colorful text

Description automatically generated        \*pLits++ = 2 \* pVars[0] + !Aig\_ObjFaninC0(pObj);

        \*pLits++ = 2 \* pVars[1] + !Aig\_ObjFaninC1(pObj);

        // negative phase

        \*pClas++ = pLits;

        \*pLits++ = 2 \* OutVar + 1;

        \*pLits++ = 2 \* pVars[0] + Aig\_ObjFaninC0(pObj);

        \*pClas++ = pLits;

ABC系统中用偶数表示本身，用大于那个偶数的基数表示取反。

Var2Lit(a):

Lit(a) = 2\*Var(a) + (是否取反1/0)

        \*pLits++ = 2 \* OutVar + 1;

        \*pLits++ = 2 \* pVars[1] + Aig\_ObjFaninC1(pObj);

    }

    // write the constant literal

    OutVar = pCnf->pVarNums[ Aig\_ManConst1(p)->Id ];

    assert( OutVar <= Aig\_ManObjNumMax(p) );

    \*pClas++ = pLits;

常数节点也转化为CNF

    \*pLits++ = 2 \* OutVar;

    // write the output literals

    Aig\_ManForEachCo( p, pObj, i )

    {

        OutVar = pCnf->pVarNums[ Aig\_ObjFanin0(pObj)->Id ];

        if ( i < Aig\_ManCoNum(p) - nOutputs )

        {

            \*pClas++ = pLits;

            \*pLits++ = 2 \* OutVar + Aig\_ObjFaninC0(pObj);

        }

        else

        {

            PoVar  = pCnf->pVarNums[ pObj->Id ];

输出节点转化为CNF

这里其实就是 同或

因为输出节点直连驱动它的节点，因此输出节点应与驱动它的节点取值相同。

            // first clause

            \*pClas++ = pLits;

            \*pLits++ = 2 \* PoVar;

            \*pLits++ = 2 \* OutVar + !Aig\_ObjFaninC0(pObj);

            // second clause

            \*pClas++ = pLits;

            \*pLits++ = 2 \* PoVar + 1;

            \*pLits++ = 2 \* OutVar + Aig\_ObjFaninC0(pObj);

        }

    }

    // verify that the correct number of literals and clauses was written

    assert( pLits - pCnf->pClauses[0] == nLiterals );

    assert( pClas - pCnf->pClauses == nClauses );

    return pCnf;

}