Binary encoding using Dichotomies

Tim Burkert and Laurenz Kamp

Contents

1 Binary encoding using Dichotomies 2

2 Implementation 3

2.1 Construction of Constrain Matrix . . . 3
2.1.1 Minimal Cover Algorithm . . . 3
2.2 Generation of Root Dichotomies . . . 3
2.3 Generation of Prime Dichotomies Candidates . . . 4
2.4 Prime Dichotomies Coverage Table . . . 4
2.5 Cover Root Dichotomies with Primes . . . 4
2.6 Optimizations . . . 4
2.6.1 Coverage Table minimization . . . 4
2.6.2 Non exact solution . . . 4

3 Evaluation 5

3.1 Runtime . . . 5
3.2 Mapping Result . . . 5
3.3 Verdict . . . 5

1 Binary encoding using Dichotomies

2 Implementation

2.1 Construction of Constrain Matrix

Before solving the input encoding problem the given problem specification must be transformed into a constrain matrix. This constrain matrix is further called A. Each row i of $A_{i,j}$ describes a constrain on the resulting encoding. Furthermore, each column j of $A_{i,j}$ assigned to a encoded symbol. For example the row $(1,1,0,0)$ means that symbols $3,4$ can be encode together so that the resulting super cube does not include symbols $1,2$. This constrain would be equal to the row $(0,0,1,1)$. The reason for this that a dichotomy describes a bipartition of a set, but for encoding problems only the relation between the elements in both sets are important. Therefore, the partition $A_0 : 1,2 A_1 : 3,4$ and the partition $B_0 : 3,4 B_1 : 1,2$ are inverse partitions but the relations are all equal. In A and B the elements $1,2$ are combined in a set and are not combined with $3,4$. A row describes also a bipartition of all symbols.

The given problem specification used symbolic names for state and binary notation including don't cares for input and output vectors. The constrain matrix A is a result of the minimal symbolic cover. Because of the binary notation of the input and output vectors we transformed the symbolic problem into a binary coded cover problem. We used the positional cube notation to deal with input and output binary notation including don't cares. For the symbolic state names we used a one hot positional encoding.

A covering super cube can for a set of positional cube notation vectors is computed by AND-conjunction all vectors. The resulting cube must be tested for validity. This is given when all cubes are valid (not equals $2'b00$).

2.1.1 Minimal Cover Algorithm

For implementation we implemented an iterative approach. The algorithm terminates when no further improvements are possible. This is when all combination of entries are tested. When a optimization is possible the two vectors that are combined are removed and a new vector covering both is included. When this happens all combinations of entries are tried again until termination.

From the set of resulting super cubes for the states constrain matrix A can be constructed. Only the entries that actual constrain the problem are for interested, therefor entries including all symbols as symbolic implicant or entries that include only one symbolic literal as implicant can be removed. Only entries that portion the symbolic state into a relation of one symbol can be combined with others not include other symbol, like the given example, are for interested.

2.2 Generation of Root Dichotomies

The generation of all root dichotomies is straight forward implementation of a sequential generator. This generator uses all rows of the constrain matrix and computes all resulting root dichotomies for each row. The number of root dichotomies are equal to the number of symbols assigned a zero in the constrain row.

2.3 Generation of Prime Dichotomies Candidates

As stated before to find a exact solution all candidates prime dichotomies must be checked. A candidate is a dichotomy where a symbols are portioned in both sets. We used a long vector with positional symbol notation where each bit is assigned to a symbol. Just by iterating all possible values all prime dichotomies candidates are reached. The inverse property of dichotomies regarding encoding is used to reduce set of candidates to the half. Because all values in the lower half have a value inverse equal in the upper half. For example a positional symbol notation vector for 5 five symbols of $2'b00110$ is equal to $2'b11001$.

2.4 Prime Dichotomies Coverage Table

Before we solve the encoding problem we need to compute which prime dichotomy covers which root dichotomies. For this we generate a table of lookup vectors. Each vector describes the cover relation between a prime dichotomy and all root dichotomies, by using again a positional notation where a bit describes the coverage.

2.5 Cover Root Dichotomies with Primes

Using the before created table it is now possible to look for a minimal set of prime dichotomies that cover all root dichotomies. We start with the small set possible of one prime dichotomy. When no solution is found the search space is increased by one additional element. The combination of elements for the minimal set of prime dichotomies is generator that iterate all combination without repetition. For the k round $\binom{n}{k}$ for n table entries are possible.

2.6 Optimizations

The complexity to find an exact solution scales with $\binom{n}{k}$, where n is the size of the coverage table and k is the amount of lines that are combined to find a solution. One way to shorten runtime is to reduce the table size. Another way is to find a faster algorithm to search the table, but this comes at the cost of non-exact solutions.

2.6.1 Coverage Table minimization

Some entries in the coverage table can be removed without lowering the result quality. A line l_1 in the table is eligible for removal if it is covered by another line l_2 in the table. l_1 is covered by l_2 if all bits that are set in l_1 are also set in l_2. Reducing the table size in this way takes additional time and is more expensive for large tables, but it can reduce table sizes drastically. Evaluation has shown that large coverage tables can be reduced by a factor of 2.

2.6.2 Non exact solution

3 Evaluation

This evaluation analysis the performance and drawbacks of an exact binary state encoding. To measure the result quality we used the ABC a tool for sequential synthesis and verification. Our implementation exports a complete finite state machine using the before computed encoding solution. As required by the minitasked abc was used to mapped the state machine onto LUTs.

3.1 Runtime

3.2 Mapping Result

3.3 Verdict

As seen in the mapping result compared to one hot encoding binary state encoding using dichotomies delivers worse result for the target technology FPGA, when area consumption is compared. For this technology a reduction for boolean function below 5 inputs doest reduce the used area, as seen in the lut description.

LUT description

For example, when using a gate array target technology the amount of used logic gates and latches is reduced compared to a one hot encoding. Reference to result table with latches

To summaries, an exact encoding algorithm should only be used if the target technology can utilize the improved solution and when the resulting enormous runtime increase is acceptable.