ARRAY CONSTRAINT TECHNOLOGY

GenoKey

## Solving Complex Problems

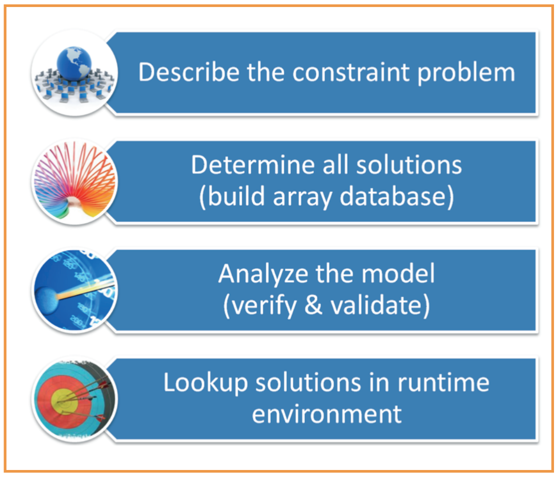
Traditional approaches to solving large scale constraint problems face a seemingly unsolvable paradox; complete resolution of all possible permutations of constraints is either impractically slow or memory consuming on standard hardware platforms for problems of real-world scale. GenoKey's patented array constraint technology satisfies *all* of these seemingly contradictory requirements at the same time:

- **Scalability**
  GenoKey's unique Array Based Constraint Engine is capable of solving large-scale problems, i.e., logical models with thousands of variables and a multitude of combinations. For example, safety critical healthcare systems routinely use more than 10,000 variables and a state space with more than $10^{300}$ combinations.
- **Completeness**
  Using GenoKey's Array Toolkit, all combinations of the entire state space are addressable to ensure that *all* constraints on *all* variables are taken into account. This is critical whenever a complete verification (consistency check) of the logical model is required.
- **Compactness**
  The solution space of the constraint problem (usually a multitude of valid combinations) can be represented in GenoKey's Array representation as a compact yet complete data model.
- **Real-time capability**
  When querying the data model, *all* constraints on *all* selected variables can be deduced in real time. The deduction can be completed with a small and predictable response time and a predictable allocation of memory. The high-performance Array Runtime Engine makes it possible to run constraint-based applications on small computers such as smartphones and/or tablets.
- **Embeddable**
  GenoKey's Array Based constraint engine can be used as a component in workflows or larger software applications

GenoKey's Array Constraint technology offers a unique solution to complex constraint problems on finite domains using limited computer resources.

While most constraint solvers are based on heuristic search algorithms, the array approach is different. GenoKey's Array Constraint technology optimizes runtime performance by indexing into a complete solution space, which has been established at compile time - it does not require interpreting rules or relations at runtime.

In array-based logic, logical constraints on finite domains (the relationships between all variables) are represented in terms of geometrical data structures (nested arrays). The geometrical approach to logic makes it possible to deduce all logical information very efficiently by simple parallel array operations rather than heuristic search algorithms. Array-based logic provides a common geometrical approach to any constraint problem on finite domains, including propositional logic, predicate logic, and relational algebra.


Describe the constraint problem
Determine all solutions (build array database)
Analyze the model (verify & validate)
Lookup solutions in runtime environment

**Four stage process for constraint satisfaction**

| | Array Based Logic Methods | Search Methods |
|---|---|---|
| Compiler | • Compile each relation<br>• Combine relations with common variables<br>• Test for logical consistency, bounded variables, etc.<br>• Find full solution space | • Compile each rule or relation<br>• Check for logical consistency |
| Runtime | • Fast and predictable processing time (real-time)<br>• Very compact data model<br>• Simple array lookups<br>• Very suitable for embedded systems | • Heuristic search until all constraints considered<br>• Unpredictable processing time on complex structures<br>• Not suitable for embedded systems |

The GenoKey technology, which is developed and patented by Dr. Gert L. Møller (co-founder of GenoKey A/S), applies the mathematical foundation of *Array-Based Logic*, which was developed in co-operation with the Technical University of Denmark.

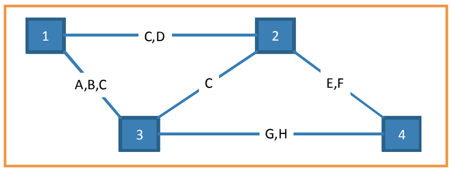
The technology is subject to various patent including U.S. Patent No 6,633,863 or European Patent No 1062603.

# 4 Stage Constraint Satisfaction

## Describe the Problem

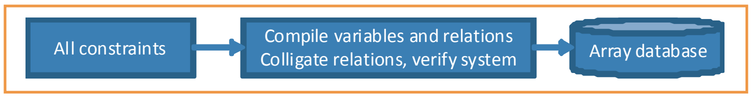
Constraints are expressed in terms of a set of relations on a predefined set of variables. The variables can be any data type on finite domains, including interval variables. A relation can be any propositional form, which can be compiled into a truth table. A relation may therefore be described in many ways; either a symbolic expression or a table representing valid or invalid combinations. This makes it easy to import and reuse data from other sources, for example an existing relational database.

Often the array model source file will count a large number of interconnected relations, i.e., relations with common variables. The colligation graph right illustrates the structure of a small system with the nodes and arcs representing relations and connecting variables, respectively. For example, the relations 1 and 2 are interconnected by the common variables C,D.

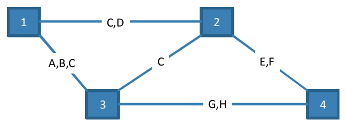
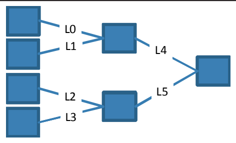


## Determine all Solutions

In order to determine the complete solution space, all invalid combinations must be eliminated.


All constraints
Compile variables and relations
Colligate relations, verify system
Array database

- Compile each relation - A relation is any kind of propositional function specifying valid and invalid combinations of a given set of variables. After compilation each relation is compressed into a very compact nested array representing the valid combinations.
- Colligate interconnected relations - The solutions (valid combinations) must satisfy all constraints, i.e., the conjunction of all relations. All relations with common variables are thus colligated (joined) in order to eliminate the invalid combinations of the common variables.
- Compress the solution space - Usually, there will be a multitude of valid combinations. The array database uses nested arrays which give a very compact representation, and any valid combination can be found by simple indexing and table look up on these arrays. Generally the solution space is represented by one or more tree structures of interconnected relations with common link variables.

| Constraint problem (before compilation) | Solution space (after compilation) |
|---|---|
|  |  |

- The structure of the constraint problem before and after compilation is sketched in the two colligation graphs above. Each user-defined variable (A,B,…G,H) will be isolated into a single relation after compilation - all connections between the new relations in the solution space will be done by the system generated link variables (L0, L1,..,L5). Thus all constraints are eliminated at compile time, not just the constraints of the isolated relations, but also the constraints of the interconnected relations.
- Many alternative tree structures can represent the solution space of a given constraint problem. The binary array database can thus be compiled with various parameters, for example in order to reduce the size or optimize the runtime performance of the database.

## Analyze the Model

Since all constraints are identified and eliminated at compile time, it is easy to perform advanced analysis of the logical model, for example:

- Check for consistency and bounded variables (verification) - if there are no solutions, the compiler can trace the relations causing the contradiction. Similarly, if some variable elements are invalid due to the system constraints, the user will be advised, e.g, if the user has defined a variable METABOLITE with the values HIGH, MEDIUM, LOW it is important to know if the value HIGH is never permitted.
- Check for redundant or superfluous information - redundant information (for example two or more relations with common logical constraints) will be eliminated automatically in the colligation process. Independent or unconstrained variables will also be identified and eliminated when the array database is generated.

## Use the Solutions

The logical model is now very easy to use, and the logical consequences of any query can be deduced very fast by simple table look up and indexing on the complete solution space. It could for example be a healthcare decision support system running on a smartphone or tablet.

- State deduction - When one or more variables are measured or selected from the environment, all constraints on all variables (completeness of deduction) can be determined by a simple state propagation on the binary array database.
- Deduction of derived relations - Any derived relation on a subset of variables can easily be deduced to check the mutual constraints on those variables.