Arc Consistency with Negative Variant Tables

Albert Haag[1]

Abstract. In this paper I discuss a fundamental difference between positive and negative variant tables (tables listing excluded combinations) from the viewpoint of a modeler. I provide an approach to achieving arc consistency with negative tables that can be integrated into an existing configurator that already implements methods for arc consistency. I also provide a simple necessary condition to test whether a further restriction of given domains is possible using a negative table. As a positive table is equivalent to a negative table representing the complement, this condition test also applies for positive tables that have a small complement. I refer to this process as *double negation*. A prototypical implementation in Java exists that covers the work described both here and in [6]. I present aggregated results of applying *double negation* to variant tables from real product test data in [6]. This also validates the overall functional correctness of the entire approach.

1 Introduction

Tabular data are an important element of product configuration models. In the context of the *SAP Variant Configurator - SAP VC* [3] a table that lists valid combinations of product properties is referred to as a *variant table*. The number of columns of a variant table is called its *arity*. Each column of the table is mapped to a product property, e.g., *Color*.

The simplest form of a product model is just as one single variant table. The term *variant table* derives from this. Table 1, below is an example of a variant table that completely describes the model of a configurable t-shirt by listing all variants. The t-shirt has three properties *Color*, *Size*, and *Print*[2]. Given the values that appear in the table[3] it is evident that 11 of possible 24 combinations have been selected as valid for configuration. I expand on this model in Section 3.

Now, SAP customers have long requested the ability to also maintain tables of disallowed (excluded) combinations of product properties. These would be called *negative variant tables*.

Whereas, a positive table implicitly defines a bound on the overall (global) domains of the associated properties[4], this is different for a negative variant table. There are at least two interpretations of the motivation for maintaining a negative table:

1. The persons maintaining a negative table are aware of overall (global) domains for the affected product properties. The negative form of the table is merely chosen as shorthand for maintaining the complement of an otherwise very large positive table.

Table 1. Variant table for a simple configurable t-shirt

Color	Size	Print
Black	Small	MIB
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

2. The persons maintaining the table are expressing exclusions completely independently of any thought of what the affected property domains might be.

One obvious way of dealing with negative variant tables that pertains to the first case is to provide support for complementing the table with respect to global domains of the product properties at maintenance time[5]. Note that it is not possible to calculate the complement to the global domains if these are unconstrained[6].

I shall focus on the second case as requests by SAP customers clearly indicate this setting. Then, it is not legal to complement a negative table at maintenance time with regard to the global domains even if these are finite, because they may change after the table was maintained. The table may have its own update cycle and should, therefore, not need to be touched each time a global domain changes.

In knowledge-based configuration variant tables function as table constraints. The extensional form of a table constraint (explicitly listing all tuples of values implied by the table) closely corresponds to the relational form of a variant table (directly storing the variant table transparently in a relational database[7]). Each value a_{ij} (where i is the index of the row, and j is the index of the column) that occurs in a table cell states a Boolean proposition p_{ij} that the corresponding mapped property v_j is assigned to that value ($p_{ij} \models (v_j = a_{ij})$). In this case, a row r_i in the variant table directly maps to a tuple of propositions τ_i in the associated table constraint representing the

[1] SAP SE, Germany, email: albert.haag@t-online.de

[2] The example is taken from [1]. I use and extend it both here and in [6]

[3] I have kept the shorthand codes of MIB (for "Men in Black") and STW (for "Save the Whales") used in [1]

[4] That is to say, for a positive table a value that does not occur in a particular table column can be removed from the domain of the property associated with that column

[5] If the challenge lies in the large size of the positive variant table, offering maintenance of the table directly in a compressed format is another option. Compressions of Table 1 are depicted in [6]. For the SAP VC some support is provided for complementing tables at maintenance time and maintaining tables in non-extensional form, i.e., with cells containing non-atomic entities such as real-valued intervals or sets of values

[6] Unconstrained domains are not that common in traditional business settings, but they do occur. For real-valued numeric properties the global domains may often be (bounded/unbounded) continuous intervals

[7] In this case, the table cells will contain only atomic values (Strings or numbers)

conjunction of its elements, i.e., for fixed row index i and arity k $r_i = (a_{i1}, \ldots, a_{ik})$ and $\tau_i = (p_{i1}, \ldots, p_{ik}) \models p_{i1} \wedge \ldots \wedge p_{ik}$.

A central form of constraint evaluation is propagation to achieve arc consistency, which removes any values from the current domains of the properties constrained by the variant table that are not supported in the intersection of the table with these domains. This is a proven way in practice to limit the choices available to the user in interactive configuration. I refer to this simply as constraint propagation in the sequel[8].

In this paper, I do not propose an own algorithm for constraint propagation with negative tables[9]. Rather, I assume that a configurator will already implement constraint propagation for positive tables, but it may not implement a corresponding algorithm (such as [9]) for negative table constraints[10].

A tuple representing a disallowed combination of propositions $(p_1, \ldots, p_k)$ in a negative variant table of arity k allows k obvious inferences:

$$p_{j_1} \wedge \ldots \wedge p_{j_{k-1}} \rightarrow \neg p_{j_k} \quad (1)$$

These could be applied at run-time if and where the configurator supports it, e.g., where it is possible to remove a value from a domain in a way that can be represented in the configurator[11].

In this paper, I characterize what can be achieved using pre-existing means of constraint propagation beyond (1) with negative tables that are maintained independently from the global domains for the mapped properties. I give a condition that can be tested during evaluation to determine if further restrictions via constraint propagation are possible (Section 4). The condition states that at least all but one of the domains must be sufficiently restricted in order to achieve further filtering with the negative variant table. In a way, this generalizes (1), which states that an inference is possible if all but one value assignments are known.

The condition test rests on the fact that the solution set of a negative variant table with arity k can be easily decomposed at run-time into $k + 1$ disjoint parts, of which k are in the form of Cartesian products (referred to as *c-tuples* in Section 2.2). This decomposition is one of the results presented in this paper and applies independently and on top of the methods implemented in the configurator for arc consistency.

A positive variant table can be transformed into a negative one by negating (complementing) it. When processing this negative table the table is logically negated again, yielding an identical solution set, but not an identical structure to the original table. I refer to this as *double negation*, and discuss it as one possible approach to compression in Section 5. Otherwise, compression of variant tables is a topic I deal with in [6].

I distinguish between information known to the modeler at the time the variant table is maintained (maintenance-time) from information known when configuring the product (run-time). Some calculations can already be performed at maintenance-time, others best at run-time. It is important that run-time operations do not impede the overall performance of the configurator. Maintenance-time operations should be performant as well, but this is less critical.

In Section 2, I introduce the notation and some formalisms. Section 3 uses the product model of a t-shirt taken from [1] (Table 1) to illustrate the concepts. As already mentioned, Sections 4 and 5 deal with deriving the necessary condition test and with double negation, respectively. In Section 6, I comment on the status of the implementation. I conclude with some further observations in Section 7.

Finally, a disclaimer: While the motivation for this work lies in my past at SAP and is based on insights and experiences with the product configurators there [3, 5], all work on this paper was performed privately during the last two years after transition into partial retirement. The implementation is neither endorsed by SAP nor does it reflect ongoing SAP development.

2 Framework

My scope here is restricted to the problem of constraint propagation with negative tables. I construct a framework for this that is based on operations with sets. After an overview of the basic framework, I define the relevant sets in Section 2.1, and present the actual framework I use for negative variant tables in Section 2.2.

A variant table $\mathcal{T}$ is characterized as follows: its *arity* expresses how many columns it has. The columns are indexed with $j : 1 \leq j \leq k$, where k is the arity. Each column is mapped to a *product property*, v_j. I assume $\mathcal{T}$ to be in relational form, i.e., each table cell contains an atomic value (a string or number). Variant tables that allow intervals or wild cards in cells are not directly representable in relational form. Many of the results here could be extended to cover this case, but I consider it out of scope here.

I view a variant table $\mathcal{T}$ in its role as a constraint only[12]. In the language of constraints the product properties mapped to the columns of the variant table are the constraint variables. I continue to refer to them as *product properties*. The assumed relational form of the variant table simplifies the correspondence between a table cell a_{ij} (where i is the index of the row, and j is the index of the column) and a value assignment $v_j = a_{ij}$. Thus each row directly corresponds to a value assignment tuple $(v_1 = a_1, \ldots, v_k = a_k)$. In its role as a constraint, each row in $\mathcal{T}$ expresses a k-tuple of valid value assignments.

Each product property, v_j, has a domain. The domain known at maintenance-time for v_j is called the *global domain*, Ω_j, which may be *unconstrained* (infinite) if unknown (or otherwise infinite as would be the case for a continuous interval). For negative variant tables, I treat global domains as *unconstrained*, as discussed in Section 1. I refer to a domain known at run-time for v_j as a *run-time restriction*, R_j, even if it also unconstrained or infinite.

I assume that the configurator already implements constraint propagation methods for arc consistency, e.g., implements some form of a GAC algorithm [2, 8]. As a consequence, I do not delve into the details of any particular GAC algorithms here[13].

The following example illustrates the concepts introduced so far, albeit for a positive variant table. Examples for negative variant tables are constructed in Section 3.

[8] I am only concerned with the propagation on each table constraint individually. The question of how to achieve overall arc consistency and how to resolve inconsistencies is up to the methods pre-implemented in the configurator. The SAP VC, for example, does not backtrack, but performs constraint propagation via forward filtering

[9] The implementation is part of a prototype for exploring the compression approach in [6]. This also handles negative variant tables, and thus an own approach is implicit in that context

[10] If it does implement such an algorithm, then the algorithm itself implicitly involves calculating the complement to the domains known at run-time

[11] This will not be the case when the run-time domain is *unconstrained*, but then all methods of dealing with negative tables referred to cannot be meaningfully applied

[12] In the SAP VC configurator variant tables are also used in procedural fashion, e.g., in "rules"

[13] If the configurator also already implements some form of arc consistency with negative table constraints such as [9], the basic approach will still apply. I shall indicate what differences must then be observed in the appropriate places, below

Example 1 *Table 1 is a positive variant table of t-shirt variants. Looking at this table, the global domains are*

- $\{Black, Blue, Red, White\}$ *for the product property "*$Color$*", denoted by* v_1
- $\{Large, Medium, Small\}$ *for the product property "*$Size$*", denoted by* v_2
- $\{MIB, STW\}$ *for the product property "*$Print$*", denoted by* v_3

If the customer wants a red t-shirt, but specifies nothing else, then the run-time restrictions are

- $R_1 = \{Red\}$ *for the product property "*$Color$*"*
- $R_2 = \{Large, Medium, Small\}$ *for the product property "*$Size$*"*
- $R_3 = \{MIB, STW\}$ *for the product property "*$Print$*"*

A GAC algorithm can then further restrict the domains of the property "$Size$*" to* $\{Large, Medium\}$ *and of the property "*$Print$*" to* $\{STW\}$*. (There are only two rows in the table involving the color "*Red*".)*

2.1 Definitions and Notation of Relevant Sets

Let $\mathcal{T}$ denote a variant table of arity k for product properties $v_1 \dots v_k$. These are the product properties that are related via $\mathcal{T}$ as a constraint, and the only properties to consider when looking at $\mathcal{T}$ in isolation, but their existence is not directly tied to $\mathcal{T}$. Let $\Omega_1 \dots \Omega_k$ be known global domains of $v_1 \dots v_k$. I define the *solution space*, $\boldsymbol{\Omega}$, as

$$\boldsymbol{\Omega} := \Omega_1 \times \ldots \times \Omega_k \tag{2}$$

Any subset of the solutions space consisting of valid tuples defines a constraint relation on $v_1 \dots v_k$. In particular, $\mathcal{S}^{\mathcal{T}}$, the set of valid tuples defined by $\mathcal{T}$, is a subset of $\boldsymbol{\Omega}$. If $\mathcal{T}$ is in relational (extensive) form, then $\mathcal{S}^{\mathcal{T}} = \mathcal{T}$ for a positive table, and $\mathcal{S}^{\mathcal{T}} = \boldsymbol{\Omega} \setminus \mathcal{T}$ for a negative table. I refer to $\mathcal{S}^{\mathcal{T}}$ as the *solution set* of $\mathcal{T}$. Generally, a solution set as defined above may not be finite.

Let $\tau = (\tau_1, \dots, \tau_k) \in \boldsymbol{\Omega}$ denote a tuple in the solution space. Given any $\boldsymbol{X} \subseteq \boldsymbol{\Omega}$, I define the j-th *column domain* of $\boldsymbol{X}$ as

$$\pi_j(\boldsymbol{X}) := \bigcup_{\tau \in \boldsymbol{X}} \{\tau_j\} \tag{3}$$

I call $\pi_j(\boldsymbol{X})$ the *projection* of $\boldsymbol{X}$ onto the j-th component, and define the *projection* or *constraint propagation* operator π as:

$$\pi(\boldsymbol{X}) := \pi_1(\boldsymbol{X}) \times \ldots \times \pi_k(\boldsymbol{X}) \tag{4}$$

The complement of a column domain $\pi_j(\boldsymbol{X})$ with respect to a run-time restriction R_j plays a central role in the decomposition in Section 2.2. Hence, I find it convenient to abbreviate its notation as:

$$\overline{\pi_j(\boldsymbol{X})}^R := R_j \setminus \pi_j(\boldsymbol{X}) \tag{5}$$

For notational convenience I refer to any set $\boldsymbol{C} \subseteq \boldsymbol{\Omega}$ that is a Cartesian product $\boldsymbol{C} = C_1 \times \ldots \times C_k$ as a *c-tuple*. All of the following are c-tuples

- $\boldsymbol{\Omega}$ itself
- $\pi(\boldsymbol{X})$ the tuple of column domains for in (4)
- the tuple of run-time restrictions, denoted by

$$R = R_1 \times \ldots \times R_k$$

I refer to R as a whole as as a *run-time restriction tuple*

In Example 1 it can be seen that it is possible to eliminate $\{Small\}$ and $\{MIB\}$ after deciding on a red t-shirt. This is the basic inference of arc consistency, which I refer to as *constraint propagation*: filtering out values that are no-longer part of any valid tuple.

Given a run-time restriction tuple R and a variant table $\mathcal{T}$ with solution set $\mathcal{S}^{\mathcal{T}}$, then the set of remaining valid tuples is $R \cap \mathcal{S}^{\mathcal{T}}$. I abbreviate the solution set $\mathcal{S}^{R \cap \mathcal{S}^{\mathcal{T}}}$ by $\mathcal{S}^{\mathcal{T},R}$. If $\mathcal{T}$ is positive, then $\mathcal{S}^{\mathcal{T},R} = \mathcal{T} \cap R$. If $\mathcal{T}$ is negative, then $\mathcal{S}^{\mathcal{T},R} = R \setminus \mathcal{T}$.

Constraint propagation restricts the run-time restrictions R_j to $\pi_j(R \cap \mathcal{S}^{\mathcal{T}})$.

2.2 Negative Variant Tables

For clarity, I denote a *negative* variant table by $\mathcal{U}$. I take $\mathcal{U}$ to be in relational form and have arity k. The complement of $\mathcal{U}$ with respect to $\pi(U)$ is a positive table constraint that can be calculated at maintenance-time and depends only on $\mathcal{U}$ itself. I denote this complement by $\overline{\mathcal{U}}$

$$\overline{\mathcal{U}} = \pi(U) \setminus \mathcal{U} \tag{6}$$

Note that $\overline{\mathcal{U}} = \emptyset$ by construction if $\mathcal{U}$ has only a single line, because then $\pi(U) = \mathcal{U}$.

Given a run-time restriction tuple R, the solution set of $\mathcal{U} \wedge R$ is $\mathcal{S}^{\mathcal{U},R} = R \setminus \mathcal{U}$. This can be decomposed into two disjoint parts (either of which may be empty, see Section 3 for examples):

$$\mathcal{S}^{\mathcal{U},R} = (R \setminus \pi(U)) \cup (\overline{\mathcal{U}} \cap R) \tag{7}$$

The first part $(R \setminus \pi(U))$ is just R with the c-tuple spanning all values that occur in $\mathcal{U}$ removed. The second part then re-adds all tuples in $\pi(\mathcal{U})$ that occur both in $\overline{\mathcal{U}}$ and R.

The second part $(\overline{\mathcal{U}} \cap R)$ is just the solution set of $\overline{\mathcal{U}} \wedge R$. As $\overline{\mathcal{U}}$ is a positive variant table, this can be processed by the means available to the configurator[14].

The first part $(R \setminus \pi(U))$ can be decomposed into k disjoint c-tuples $\boldsymbol{C}_j^{\mathcal{U},R}$ as follows (see Proposition 2):

$$\begin{aligned}
\boldsymbol{C}_1^{\mathcal{U},R} &= \overline{\pi_1(\mathcal{U})}^R \times R_2 \times R_3 \times \ldots \times R_k \\
\boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \overline{\pi_2(\mathcal{U})}^R \times R_3 \times \ldots \times R_k \\
&\cdots \\
\boldsymbol{C}_k^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \pi_3(\mathcal{U}) \times \ldots \times \overline{\pi_k(\mathcal{U})}^R
\end{aligned} \tag{8}$$

or more explicitly for the omitted rows $2 < j < k$:

$$\boldsymbol{C}_j^{\mathcal{U},R} = (\pi_1(\mathcal{U}) \times \ldots \times \pi_{j-1}(\mathcal{U})) \times \overline{\pi_j(\mathcal{U})}^R \times (R_{j+1} \times \ldots \times R_k)$$

Proposition 2 $(R \setminus \pi(\mathcal{U}))$ *in* (7) *can be decomposed into the disjoint union*[15] (8) *for an arbitrary ordering of the columns.*

Proof
Each $\boldsymbol{C}_j^{\mathcal{U},R}$ is a subset of R by construction.
The j-th component of $\boldsymbol{C}_j^{\mathcal{U},R}$ is $\overline{\pi_j(\mathcal{U})}^R$. This is disjoint to $\pi_j(\mathcal{U})$. So $\boldsymbol{C}_j^{\mathcal{U},R}$ is disjoint to all $\boldsymbol{C}_p^{\mathcal{U},R} \quad \forall j < p \leq k$.
No tuple $x \in \pi(\mathcal{U})$ is in any $\boldsymbol{C}_j^{\mathcal{U},R}$, because $x \in \boldsymbol{C}_j^{\mathcal{U},R} \Rightarrow x_j \in \overline{\pi_j(\mathcal{U})}^R \Rightarrow x_j \notin \pi_j(\mathcal{U}) \Rightarrow x \notin \pi(\mathcal{U})$.

[14] If the configurator implements an algorithm for negative GAC such as [9], then $\overline{\mathcal{U}}$ need not be explicitly calculated. Processing would not be affected for this part

[15] I exclusively use the term *disjoint union* to refer to a union of disjoint sets [4]. I denote the disjoint union of two sets A and B by $A \cup B$, which implies that $A \cap B = \emptyset$

For all other tuples $y \in R$, there is at least one component with $y_j \notin \pi_j(\mathcal{U})$. Let j^* denote the smallest such index. Then, it follows by construction that $y \in \boldsymbol{C}_{j^*}^{\mathcal{U},R}$, because $\forall p < j^* : y_p \in \pi_p(\mathcal{U})$, $y_{j^*} \in \overline{\pi_{j^*}(\mathcal{U})}^R$, and $\forall p > j^* : y_p \in R_p$. ∎

The decomposition (8) is meaningful, because if $(R \setminus \pi(\mathcal{U}))$, the first part in (7), does not allow further constraint propagation, then the GAC algorithm need not be applied for $\overline{\mathcal{U}} \cap R$ in the second part at all. (All values in R are allowed by the first part.) I give a simple criteria for when this is the case in Section 4. This is a necessary precondition for a further reduction. Generally, it is easy to perform constraint propagation on a constraint that is a c-tuple[16].

3 Examples

I base my examples on the t-shirt model I introduced in Table 1. The global domains are listed in Example 1. In a real application setting, Table 1 would be used as the product model as is. For purposes of exposition here, I assume that the model evolves over time and further colors, sizes, and prints might be added in later model updates along with associated constraints. The restriction to the initial 11 valid tuples in Table 1 implements constraints to the effect that MIB implies $Black$ and STW implies $\neg Small$.

In this section, I use $\mathcal{U}$ to denote a negative variant table, and $\mathcal{T}$ to denote a positive one.

3.1 T-Shirt Example: STW

The constraint $STW \rightarrow \neg Small$ in the t-shirt example can be formulated as a single exclusion, yielding a negative variant table $\mathcal{U}$ with one row, $k = 2$, $v_1 = Print$, and $v_2 = Size$

$$\mathcal{U} = \begin{pmatrix} STW & small \end{pmatrix}$$

As noted, $\overline{\mathcal{U}} = \emptyset$.

Assume that the restricted domains at run-time are just the global domains, i.e., $R_1 = \Omega_1$ (property $Print$) and $R_2 = \Omega_2$ (Property $Size$). Then, the solution set $\mathcal{S}^{\mathcal{U},R}$ is directly given by the decomposition (8) of the first term in (7)[17]. This means that $\pi_1(\mathcal{U}) = \{STW\}$, $\pi_2(\mathcal{U}) = \{Small\}$ and:

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} r &= \{MIB\} \times \{Large, Medium, Small\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \{STW\} \times \{Large, Medium\} \end{aligned} \tag{9}$$

The solution set is the five tuples in (9):

$$(MIB, Large), (MIB, Medium), (MIB, Small),$$
$$(STW, Large), (STW, Medium)$$

Constraint propagation does not produce a domain reduction (verifiable by inspection).

If, instead, the domain restriction for v_2 at run-time is $R_2 = \{Small\}$ (but still $R_1 = \Omega_1$), then

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{MIB\} \times \{Small\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \emptyset \quad (= \{STW\} \times (\{Small\} \setminus \{Small\})) \end{aligned}$$

The solution set is now the tuple $(MIB, Small)$. Constraint propagation produces a domain reduction of R_1 to $\{MIB\}$.

[16] Constraint propagation on a solution set in the form of a c-tuple means intersecting the c-tuple with the given run-time restriction tuple R. In a product configuration context the values will most often be ordered. Hence, set operations can use binary search and are relatively efficient

[17] The unaffected property *Color* could take any value. I take this up in Section 3.5

3.2 T-Shirt Example: MIB

The constraint $MIB \rightarrow Black$ in the t-shirt example could be formulated as three exclusions against the original global domain of the property *Color* yielding a negative variant table with three rows, $k = 2$, $v_1 = Print$, and $v_2 = Color$:

$$\mathcal{U} = \begin{pmatrix} MIB & Red \\ MIB & White \\ MIB & Blue \end{pmatrix}$$

Using the global domains given in Example 1, this means that $\pi_1(\mathcal{U}) = \{MIB\}$ and $\pi_2(\mathcal{U}) = \{Red, White, Blue\}$:

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{STW\} \times \{Black, Red, White, Blue\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \{MIB\} \times \{Black\} \end{aligned} \tag{10}$$

It still holds that $\overline{\mathcal{U}} = \emptyset$. So the solution set is still directly given by the first component in (7) and its decomposition in (8), and it follows from (10) that there are five tuples in the solution set. Again, constraint propagation does not produce a domain reduction.

If, instead, the domain restriction for v_2 at run-time is $R_2 = \{Red, Blue\}$ (but still $R_1 = \Omega_1$), then

$$\begin{aligned} \boldsymbol{C}_1^{\mathcal{U},R} &= \{STW\} \times \{Red, Blue\} \\ \boldsymbol{C}_2^{\mathcal{U},R} &= \emptyset \quad (= \{MIB\} \times (\{Red, Blue\} \setminus \pi_2(\mathcal{U}))) \end{aligned}$$

The solution set $\mathcal{S}^{\mathcal{U},R}$ is now two tuples (STW, Red) and $(STW, Blue)$. Constraint propagation produces a domain reduction of R_1 to $\{STW\}$

3.3 Original T-Shirt Example in a Single Negative Table $\mathcal{U}$

In the sequel, the order of the complete set of constraint variables is $v_1 = Color$, $v_2 = Size$, $v_3 = Print$.

Let T be the variant table of the t-shirt in its original positive form given in Table 1. $\mathcal{T}$ has 11 solutions out of a possible 24. Thus complementing the table with respect to the global domains in Example 1 yields a negative table $\mathcal{U}$ with thirteen rows, $k = 3$, and

$$\mathcal{U} = \begin{pmatrix} Black & Small & STW \\ Red & Small & MIB \\ Red & Medium & MIB \\ Red & Large & MIB \\ Red & Small & STW \\ White & Small & MIB \\ White & Medium & MIB \\ White & Large & MIB \\ White & Small & STW \\ Blue & Small & MIB \\ Blue & Medium & MIB \\ Blue & Large & MIB \\ Blue & Small & STW \end{pmatrix}$$

Let $R_1 = \Omega_1$ ($Color$), $R_2 = \Omega_2$ ($Size$), and $R_3 = \Omega_3$ ($Print$). $\pi_j(\mathcal{U}) = R_j$ for every j, therefore all $\overline{\pi_j(\mathcal{U})}^R = \emptyset$, and all $\boldsymbol{C}_j^{\mathcal{U},R} = \emptyset$. In this case, $\mathcal{T} = \overline{\mathcal{U}}$, and the tuples in Table 1 are just the solution set.

3.4 Extending the T-Shirt Model

3.4.1 Extending the Global Domains

First, assume that after $\mathcal{U}$ in Section 3.3 has been maintained, the global domains of the properties are extended – without adding any exclusions ($\mathcal{U}$ remains unchanged)[18] – as follows:

- $Yellow$ and $DarkPurple$ are added to Ω_1 ($Color$)

$$\Omega_1 = \{Black, Red, White, Blue, Yellow, DarkPurple\}$$

- XL and XXL are added to Ω_2 ($Size$)

$$\Omega_2 = \{Large, Medium, Small, XL, XXL\}$$

- $none$ is added to Ω_3 ($Print$)

$$\Omega_3 = \{MIB, STW, none\}$$

Let the run-time domain restrictions reflect the changed global domains $R_1 = \Omega_1$, $R_2 = \Omega_2$, and $R_3 = \Omega_3$. Since $\mathcal{U}$ is not changed, $\pi(U)$ does not change either. It still holds that

$$\begin{aligned}\pi_1(\mathcal{U}) &= \{Black, Red, White, Blue\}\\ \pi_2(\mathcal{U}) &= \{Large, Medium, Small\}\\ \pi_3(\mathcal{U}) &= \{MIB, STW\}\end{aligned}$$

Hence, $\overline{\mathcal{U}}$ is not changed either (still equal to Table 1), and with

$$\begin{aligned}\overline{\pi_1(\mathcal{U})}^R &= \{Yellow, DarkPurple\}\\ \overline{\pi_2(\mathcal{U})}^R &= \{XL, XXL\}\\ \overline{\pi_3(\mathcal{U})}^R &= \{none\}\end{aligned}$$

(8) now yields

$$\begin{aligned}\boldsymbol{C}_1^{\mathcal{U},R} &= \{Yellow, DarkPurple\} \times R_2 \times R_3\\ \boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \{XXL, XL\} \times R_3\\ \boldsymbol{C}_3^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \{none\}\end{aligned} \tag{11}$$

In this example, both components $R \setminus \pi(\mathcal{U})$ and $\overline{\mathcal{U}} \cap R$ in (7) are non-empty and contribute to the solution set $\mathcal{S}^{\mathcal{U},R}$.

Note that $|R| = 6 \times 5 \times 3 = 90$, and (looking at (11)) the total number of solutions s for $\mathcal{U}$ is

$$s = |\boldsymbol{C}_1^{\mathcal{U},R}| + |\boldsymbol{C}_2^{\mathcal{U},R}| + |\boldsymbol{C}_3^{\mathcal{U},R}| + |\overline{\mathcal{U}}| = 30 + 24 + 12 + 11 = 77$$

3.4.2 Extending the Constraints

If product management notices that $Yellow$ does not go with MIB, then corresponding exclusions must be added to $\mathcal{U}$. This can be done in several ways. In this example, I assume that $Yellow$ and the corresponding exclusions are added before any of the other changes to the domains are made[19]. Then, the global domain given in Example 1 for the product property $Color$, denoted by Ω_1, is augmented by the value $Yellow$, and Ω_2 ($Size$) and Ω_3 ($Print$) remain unchanged. The following exclusions must be added to $\mathcal{U}$ in 3.3:

$$\begin{aligned}&\neg(Yellow, Small, MIB)\\ &\neg(Yellow, Medium, MIB)\\ &\neg(Yellow, Large, MIB)\\ &\neg(Yellow, Small, STW)\end{aligned}$$

In this situation

$$\pi_1(\mathcal{U}) = \{Yellow, Black, Red, White, Blue\}$$

changes, and $\overline{\mathcal{U}}$ changes accordingly. Two values are added to allow the color of $Yellow$ in sizes $Large$ and $Medium$ with the print STW. So $|\overline{\mathcal{U}}| = 13$. (11) holds with the modified version of $\pi_1(\mathcal{U})$. Again, both components in (7) contribute to the solution set $\mathcal{S}^{\mathcal{U},R}$, and after adding all remaining new values

$$\begin{aligned}\boldsymbol{C}_1^{\mathcal{U},R} &= \{DarkPurple\} \times R_2 \times R_3\\ \boldsymbol{C}_2^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \{XXL, XL\} \times R_3\\ \boldsymbol{C}_3^{\mathcal{U},R} &= \pi_1(\mathcal{U}) \times \pi_2(\mathcal{U}) \times \{none\}\end{aligned} \tag{12}$$

As above, $|R| = 6 \times 5 \times 3 = 90$. Looking at (12), the total number of solutions s for $\mathcal{U}$ is now

$$s = 15 + 30 + 15 + 13 = 73$$

(The four new exclusions are subtracted from the solutions in Section 3.4.1)

3.5 Excursion on Table Compression

From Sections 3.1 and 3.2 it is clear that the t-shirt table can be expressed in much more compact form by looking at the two constraints in two individual negative variant tables than at the single table in Section 3.3. In both Sections 3.1 and 3.2, the solution set is defined only by the decomposition of $(R \setminus \pi(U))$ into c-tuples. An overall solution set can be obtained by expanding these solution sets to account for the respective unconstrained property, and then intersecting the resulting two expanded solution sets.

Let the properties be ordered as $v_1 = Color$, $v_2 = Size$, and $v_3 = Print$, and the solution space $\boldsymbol{\Omega}$ given by the global domains in Example 1. In Section 3.1 the property $v_1 = Color$ is unconstrained. Set $\pi_1(\mathcal{U}) = \Omega_1$ (which implies $\overline{\pi_1(\mathcal{U})}^R = \emptyset$ for all $R \subset \boldsymbol{\Omega}$). Then, the decomposition of $R \setminus \pi(U)$ in (9) now results in three c-tuples $C'_{j(STW)}$ (one of them empty by construction):

$$\begin{aligned}C'_{1(STW)} &:= \emptyset \quad (= \overline{\pi_1(\mathcal{U})}^R \times \Omega_2 \times \Omega_3)\\ C'_{2(STW)} &:= \Omega_1 \times \{Large, Medium\} \times \Omega_3\\ C'_{3(STW)} &:= \Omega_1 \times \{Small\} \times \{MIB\}\end{aligned} \tag{13}$$

Similarly, for Section 3.2 (10) can be replaced by:

$$\begin{aligned}C'_{1(MIB)} &:= \{Black\} \times \Omega_2 \times \Omega_3)\\ C'_{2(MIB)} &:= \emptyset \quad (= \{Black\} \times \overline{\pi_2(\mathcal{U})}^R \times \Omega_3)\\ C'_{3(MIB)} &:= \{Red, White, Blue\} \times \Omega_2 \times \{STW\}\end{aligned} \tag{14}$$

The solution set $\mathcal{S}^{\mathcal{T}}$ of the original positive $\mathcal{T}$ given in Table 1 is

[18] Leaving $\mathcal{U}$ unchanged implicitly changes the underlying positive constraints. It now no-longer holds that $MIB \rightarrow black$, This is taken to be an intended consequence of using a negative variant table in a product model

[19] This assumption is made to keep the example simple. The general problem of achieving a good compression directly from a positive table is addressed in [6]

the intersection of the solution sets given by (13) and (14):

$$\begin{aligned}\mathcal{S}^{\mathcal{T}} = \quad & C'_{2(STW)} \cap C'_{1(MIB)} \quad \cup \\ & C'_{2(STW)} \cap C'_{3(MIB)} \quad \cup \\ & C'_{3(STW)} \cap C'_{1(MIB)} \quad \cup \\ & C'_{3(STW)} \cap C'_{3(MIB)}\end{aligned}$$

$$\begin{aligned}= \quad & \{Black\} \times \{Large, Medium\} \times \Omega_3 \quad \cup \\ & \{Red, White, Blue\} \times \{Large, Medium\} \times \{STW\} \quad \cup \\ & \{Black\} \times \{Small\} \times \{MIB\} \quad \cup \\ & \emptyset \quad (= \{Red, White, Blue\} \times \{Small\} \times \emptyset)\end{aligned}$$

The 11 tuples of $\mathcal{T}$ are represented as 3 c-tuples. The complexity of this compares favorably with the MDD representation in [1] for the same example. The representation also compares favorably with the compression of the table to *c-nodes* introduced in [7] if a suitable heuristic is applied. Thus, it should be possible to recover this compact representation from the full table. This is a topic of [6].

4 Arc Consistency for Negative Variant Tables

Let a negative table $\mathcal{U}$ of arity k and a finite run-time restriction tuple R be given. (I assume in the sequel that $\overline{\pi_j(\mathcal{U})}^R$ is finite for all columns.)

One approach at arc consistency with $\mathcal{U}$ is to use (7) directly at run-time to discard any tuples in $\mathcal{U}$ from R constructing $\pi(\mathcal{S}^{\mathcal{U},R})$ at the same time. The STR-Negative algorithm [9] is such an algorithm.

Here, I propose an alternative approach based on (7) and the decomposition (8). As already noted, a further restriction of R is only possible if the c-tuples in (8) allow a restriction. The lemma and its corollary below provide a simple necessary condition for this.

Lemma 3 *If $\overline{\pi_p(\mathcal{U})}^R \neq \emptyset$ for some column with index p, then $\forall j \neq p : \pi_j(R \cap \mathcal{S}^{\mathcal{U}}) = R_j$, i.e., no further reduction of any of the other domains is possible by constraint propagation using $\mathcal{U}$.*

Proof Suppose that the premise of the lemma holds. Without loss of generality, sort the columns such that $p = 1$. Then,

$$\boldsymbol{C}_1^{\mathcal{U},R} = \overline{\pi_1(\mathcal{U})}^R \times R_2 \times R_3 \times \ldots \times R_k$$

Any value in $\overline{\pi_1(\mathcal{U})}^R \neq \emptyset$ supports all values in R_j for $j \geq 2$. ∎

This has a trivial but important consequence:

Corollary 4 *If $\overline{\pi_j(\mathcal{U})}^R \neq \emptyset$ for more than one column, then no reduction of domains is possible by constraint propagation using $\mathcal{U}$.*

Lemma 3 and Corollary 4 generalize (1) to state that a reduction via constraint propagation is possible, if at least all but one of the domains are sufficiently restricted at run-time so that they lie within the range of $\pi(\mathcal{U})$, i.e., $\overline{\pi_j(\mathcal{U})}^R = \emptyset$.

Recall that $\pi(\mathcal{U})$ is determined when maintaining the variant table, whereas R is only known at run-time.

The examples in Section 3 illustrate that either both or only one of the components in (7) need to be considered at run-time. Let a negative table $\mathcal{U}$ of arity k and a run-time restriction tuple R be given. Then, three cases can happen:

1. $\overline{\mathcal{U}} = \emptyset$. In this case, arc consistency is obtained solely by computing the decomposition (8) of k c-tuples at run-time. Constraint propagation is achieved through directly intersecting these with R in a way that quickly tests the precondition of Lemma 3 and Corollary 4.
2. $R \setminus \pi(U) = \emptyset$. In this case, the GAC algorithm implemented in the configurator is applied to $\overline{\mathcal{U}}$ (or a GAC-negative algorithm is applied to $\mathcal{U}$ if this seems better).
3. Both components $(R \setminus \pi(U))$ and $\overline{\mathcal{U}}$ are non-empty. In this case, the component $(R \setminus \pi(U))$ is decomposed and processed as in the first case, testing the pre-conditions in Lemma 3 and Corollary 4 at the same time. The constraint propagation methods of the configurator need to be applied to $\overline{\mathcal{U}}$, if this is still indicated after processing the first part.

5 Double Negation of a Positive Variant Table $\mathcal{T}$

In the sequel, I take $\mathcal{T}$ to be a positive table. I denote the negation of $\mathcal{T}$ as the negative table $\neg\mathcal{T} := \pi(\mathcal{T}) \setminus \mathcal{T}$. Trimming any run-time restrictions to $\pi(\mathcal{T})$, $\neg\mathcal{T}$ (as a negative table) and $\mathcal{T}$ have the same solution space, and constraint propagation on $\mathcal{T}$ can be replaced by constraint propagation on $\neg\mathcal{T}$. The approach seems advantageous, if $\neg\mathcal{T}$ has a decomposition of the solution set (7) with a non-empty first part (8), i.e., if $\overline{\neg\mathcal{T}}$ is smaller than $\mathcal{T}$. I refer to this approach as *double negation*.

As an example, consider the extended model of the t-shirt as specified in Section 3.4.2. A representation as a positive table $\mathcal{T}$ has 73 tuples (rows) as inidicated there. Negating $\mathcal{T}$ against the extended global domains yields a table $\neg\mathcal{T}$ with 17 rows. $\neg\mathcal{T}$ is just the table $\mathcal{U}$ of Section 3.4.2 (and thus $\overline{\mathcal{U}}$ there corresponds to $\overline{\neg\mathcal{T}}$ here). Thus, as outlined there, $\overline{\neg\mathcal{T}}$ has a decomposition as in (7). It has 13 rows and the c-tuples indicated in (12).

Double negation is not only an approach at compression for a table with a small complement, but also allows applying Lemma 3 and Corollary 4 as a simple test, before actually evaluating the remaining doubly negated table $\overline{\neg\mathcal{T}}$. The process could be iterated, i.e., $\overline{\neg\mathcal{T}}$ could be doubly negated in turn. A fixed point occurs if $\mathcal{T} = \overline{\neg\mathcal{T}}$.

6 Implementation/Work in Progress

I have implemented the approach for treating negative tables I describe here, including *double negation*, within a Java prototype addressing the greater context of compressing and compiling variant tables [6].

As customers so far have not had the opportunity of maintaining negative tables directly in product configuration models, I have no real data to evaluate this approach. Experiences are currently limited to testing for functional correctness. Besides testing with exemplary tables such as variations of the t-shirt, I have applied the *double negation* approach to 238 SAP VC variant tables. These tables are also the basis for the evaluation of the approach at compression in [6].

Complementing a sparse table is not feasible if the solution space is large. The implementation performs complementation on a representation I term a *Variant Decision Diagram − VDD*, and produces a result in a compressed form (see [6]). I have so far limited attempts at double negation to those of the 238 tables where the number of tuples of the complement is smaller than the number of rows (tuples) in the original (relational) table. The result is given in Table 2. All tables where this criterium does not apply are counted as *Skipped*. Of the remaining tables where double negation was attempted, those where a reduction was realized (i.e., the VDD of $\mathcal{T}$ was larger than

that of $\overline{\neg\mathcal{T}}$) are counted as *Reduced*. The total number of tables in the model and the number of those neither skipped nor reduced are given for completeness.

I give more detailed results of tests with the implementation in [6]. As pointed out there, it is not yet clear whether double negation yields a gain over the general compression approach.

Table 2. Tables amenable to double negation

Number of tables			
Total	Skipped	Reduced	Not reduced
238	181	39	18

7 Conclusion

I presented an approach to handling configuration constraints defined by negative variant tables that can be integrated as an add-on to an existing configurator, such as the SAP VC, with little risk. The original mechanisms and architecture of the configurator are not touched.

SAP customers specifically request that a constraint for a negative table not be sensitive to subsequent changes to the global domains of the affected product properties. This approach meets that requirement. As the footnote to the example in Section 3.4.1 points out, this may mean that some implicit positive constraints, valid before a change to the global domains, may no-longer be valid afterwards.

This emphasizes a modeling aspect of the problem: Is it feasible and beneficial to offer the option of negative variant tables as part of the modeling environment in this fashion? This may be a tricky question. The implemented prototype offers a means for experimenting with the functionality, and thus a basis for discussing the merits and demerits of negative variant tables.

One main idea, implicit in the approach, is to make use of the distinction between information known at maintenance-time to the modeler (the table content) and information known at run-time to the configurator (the current domain restrictions). For a negative table $\mathcal{U}$ the finite column domains $\pi_j(\mathcal{U})$ can be determined at maintenance time. For a given run-time restriction of the domains R the complements of the column domains to R (denoted by $\overline{\pi_j(\mathcal{U})}^R$) must be calculated at run-time. This calculation is efficient, as the required set-operations can make use of the natural order imposed on the domain values by the business context.

If more than one $\overline{\pi_j(\mathcal{U})}^R$ is non-empty at run-time, constraint propagation does not yield a further restriction of the run-time domains. If one $\overline{\pi_j(\mathcal{U})}^R$ is non-empty, then only the run-time restriction for that column can be further restricted. When a restriction is possible by constraint propagation, the GAC algorithm only needs to be applied to the positive table $\overline{\mathcal{U}}$, the complement of $\mathcal{U}$ to $\pi(\mathcal{U})$. The remaining part of the solution set of $\mathcal{U}$ is the set-difference between two c-tuples (Cartesian products). I showed in Proposition 2 that such a set-difference can be decomposed into k disjoint c-tuples, where k is the arity of $\mathcal{U}$. Constraint propagation on c-tuples is again based on set operations that are assumed to be efficient, given the value ordering on the domains.

Of course, the approach also applies to the general case that negative tables are merely meant as a short-cut to maintaining an otherwise lengthy positive variant table.

The approach can also be applied to a positive variant table $\mathcal{T}$ by negating it, and treating its complement $\neg\mathcal{T}$ as a negative table. In this process $\neg\mathcal{T}$ is negated again ($\overline{\neg\mathcal{T}}$), which I refer to as *double negation*. The purpose of double negation is that it may yield a beneficial (partial) compression of the table, i.e., the table $\overline{\neg\mathcal{T}}$ may be smaller than $\mathcal{T}$, with the remaining part of the solution set being a set of k c-tuples that additionally allows testing whether constraint propagation is possible at all for a given run-time restriction R.

ACKNOWLEDGEMENTS

I would like to thank all that took the time to comment on previous versions of this paper, and have thus contributed to its current form. This includes the anonymous reviewers, but also colleagues at SAP, particularly Conrad Drescher and Andreas Krämer. I tired to incorporate all their suggestions. Any remaining flaws and dis-improvements are solely my responsibility.

REFERENCES

[1] H.R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *J. Artif. Intell. Res. (JAIR)*, **37**, 99–139, (2010).
[2] C. Bessiere, 'Constraint propagation', in *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, chapter 3, Elsevier, (2006).
[3] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration with SAP, second edition*, SAP Press, Galileo Press, 2012.
[4] K. Ferland, *Discrete Mathematics*, Cengage Learning, 2008.
[5] A. Haag, 'Chapter 27 - product configuration in sap: A retrospective', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 319 – 337, Morgan Kaufmann, Boston, (2014).
[6] A. Haag, 'Column oriented compilation of variant tables', in *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015.*, pp. 89–96, (2015).
[7] G. Katsirelos and T. Walsh, 'A compression algorithm for large arity extensional constraints', in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, ed., Christian Bessiere, volume 4741 of *Lecture Notes in Computer Science*, pp. 379–393. Springer, (2007).
[8] C. Lecoutre, 'STR2: optimized simple tabular reduction for table constraints', *Constraints*, **16**(4), 341–371, (2011).
[9] Honbo Li, Yanchun Liang, Jinsong Guo, and Zhanshan Li, 'Making simple tabular reductionworks on negative table constraints', in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, eds., Marie desJardins and Michael L. Littman. AAAI Press, (2013).