Interactive Configuration with Regular String Constraints

Esben Rune Hansen[1] and Henrik Reif Andersen[1]

[1] IT University of Copenhagen,
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
{esben,hra}@itu.dk

Abstract

In this paper we present a generalization of the problem of interactive configuration. The usual interactive configuration problem is the problem of, given some variables on small finite domains and an increasing set of assignment of values to a subset of the variables, to compute for each of the unassigned variables which values in its domain that participate in some solution for some assignment of values to the other unassigned variables.

In this paper we consider how to extend this scheme to handle infinite regular domains using string variables and constraints that involves regular-expression checks on the string variables. We first show how to do this by using one single DFA. Since this approach is vastly space consuming, we construct a data structure that simulates the large DFA and is much more space efficient. As an example a configuration problem on n string variables with only one solution in which each string variable is assigned a value of length k the former structure will use $\Omega(k^n)$ space whereas the latter only need $O(kn)$. We also show how this framework can be combined with the recent BDD techniques to allow both boolean, integer and string variables in the configuration problem.

Introduction

Interactive configuration is a special Constraint Satisfaction Problem (CSP), where a user is assisted in configuration by interacting with a *configurator* – a computer program. In configuration the user repeatedly chooses an unassigned variable and assigns it a value until all variables are assigned. The task of the configurator is to state the valid choices for each of the unassigned variable during the configuration. The set of valid choices for an unassigned variable x is denoted the *valid domain* of x (Hadzic *et al.* 2004) .

As an example consider the problem of assigning values to the variables x_1, x_2 and x_3 where $x_1 \in \{1, .., 5\}$ and $x_2, x_3 \in \{1, .., 10\}$ with the requirement that $x_1 = 1 \lor x_1 = 2 \lor x_2 = 2$ and $x_2 = x_3$. Initially the user can choose to assign a value from $\{1, .., 5\}$ to x_1 or assign a value from $\{1, .., 10\}$ to x_2 or x_3. Suppose now the user assigns 3 to x_3. In this case the valid domain of x_2 is $\{3\}$ and the valid domain of x_1 is $\{1, 2\}$.

The valid domain of each unassigned variable has to be updated every time a value is assigned to a variable as the assignment might make other assignments invalid as in the example above. The user interaction with the configurator has to be real-time which in practice means that the configurator has to update the valid domains within 250 milliseconds (Raskin 2000). Calculating the valid domains is NP-hard since it can be used to solve 3SAT. However if we have constructed a *binary decision diagram* (BDD) (Bryant 1986) that represents the constraints, we are able to keep the computation time polynomial in the size of the BDD. The BDD constructed can be exponentially large, but in practice BDDs have proved themselves to be far from exponential in size for many configuration problems (Hadzic *et al.* 2004).

As BDDs use binary variables to encode the domains of the variables they rely on, that these domains are small. In this paper we will consider the case of variables that take strings as their values, which implies that their domains might not be finite. Therefore the standard BDD approach will not be able to handle the problem.

As an example on the functionality we want to provide, suppose that a user has to fill in a form where there is a lot of constraints on the input. Consider a CSP with the variables phone, country, zip and district along with the following constraints:

I The prefix of phone is "+45" $\iff$ country = "Denmark"

II country = "Denmark" $\implies$ zip has four digits

III zip = "2300" $\land$ country = "Denmark" $\iff$ district = "Copenhagen S"

Suppose in the CSP above that the user entered district = "Copenhagen S". This restricts the valid domain of zip to the singleton set {"2300"} and the valid domain of country to {"Denmark"} by (III). The valid domain of phone is decreased to the set of strings which has "+45" as a prefix by (I).

Suppose instead that the user has entered phone = "+45 23493844". This decreases the valid domain of country to {"Denmark"} by (I), and the valid domain of zip to strings consisting of 4 digits. Actually this restriction will be performed as soon as the user have entered "+45", since every completion of phone achieved by appending additional letters at the end of phone still will have "+45" as a prefix.

Related Work

As mentioned in the introduction, interactive configuration of variables with finite integer domains can be done by compiling a BDD prior to the user interaction (Hadzic *et al.* 2004). It can also be done by compiling an acyclic DFA (Amilhastre, Fargier, & Marquis 2002), but since this solution is very similar to the solution using a BDD we will, for simplicity, in this paper only refer to interactive configuration based on BDDs.

The idea of using regular expressions as constraints on variables has been considered from two different perspectives by (Pesant 2004) and (Golden & Pang 2003)

In (Pesant 2004) regular expressions are applied as a global constraint to the variables in the constraint network where each variable is considered as a letter and the alphabet corresponds to the domains of the variables. Hence Pesant is considering regular expression constraints applied to one string of some fixed size where a subset of the characters in the string are assigned to values. In our case we are considering a set of strings unbounded in length, that is updated by appending characters at the end of a string. In short we are considering a much more expressive constraint language on variables with infinite domains.

In (Golden & Pang 2003) the domains of string variables are being specified by regular string constraints. In this paper the length of the strings is not bounded. However they are only considering whether or not the string variables satisfies a set of regular constraints. They do not consider mixing regular constraints and logic as we do in the current paper.

Preliminaries

Consider a CSP stated as $\mathcal{C} = (\mathcal{X}, \Sigma, \mathcal{F})$. By $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ we denote the variables of the problem. By Σ we denote an alphabet. By $\mathcal{F} = \{f_1, \ldots, f_d\}$ we denote formulas written using the syntax

$$f ::= f \vee f \mid \neg f \mid \text{match}(x, \alpha)$$

where α is a regular expression over Σ and $x \in \mathcal{X}$. The expression $\text{match}(x, \alpha)$ is true iff $x \in L(\alpha)$, where $L(\alpha)$ is the language defined by the regular expression α. We use $f \wedge g, f \Rightarrow g$ and $f \Leftrightarrow g$ as shortcuts for $\neg(\neg f \vee \neg g), \neg f \vee g$ and $(f \Rightarrow g) \wedge (g \Rightarrow f)$ respectively.

Regular expressions are written using the syntax

$$\alpha ::= \alpha\alpha \mid \alpha|\alpha \mid \alpha * \mid w$$

listed in increasing order of strength of binding, where $w \in \Sigma$.

We denote by $\rho = \{(x_1, w_1), \ldots, (x_n, w_n)\}$ a complete assignment of the values $w_1, \ldots, w_n \in \Sigma^*$ to the variables $x_1, \ldots, x_n$ that is all the variables in $\mathcal{X}$. The set of solutions to $\mathcal{C}$ is the set of assignments to $\mathcal{X}$ that satisfy all formulas in $\mathcal{F}$, that is:

$$sol(\mathcal{C}) = \{\rho \mid \rho \models f \text{ for all } f \in \mathcal{F}\}.$$

Definition 1 (Valid Domains). *The valid domain of a variable $x_k \in \mathcal{X}$ relative to an assignment ρ, denoted $V_{x_k}^\rho$, is the set of values $w \in \Sigma^*$ for which appending w to the current assignment to x_k can be extended to a solution to $\mathcal{C}$ by appending appropriate words from Σ^* to the values assigned to the remaining variables $\mathcal{X} \setminus \{x_k\}$. Stated formally:*

$$V_{x_k}^\rho = \{w \in \Sigma^* \mid \exists \rho' : \rho'(x_k) = w \wedge \rho\rho' \in sol(\mathcal{C})\}$$

where ρ and ρ' are assignments to $\mathcal{X}$ and the concatenation $\rho\rho'$ is defined by $\rho\rho' = \{(x_1, \rho(x_1)\rho'(x_1)), \ldots, (x_n, \rho(x_n)\rho'(x_n))\}$.

Theorem 1. *For any $x \in \mathcal{X}$ and any assignment ρ to $\mathcal{X}$ it holds that V_x^ρ is a regular language.*

We will prove this theorem by showing how we for any given variable $x_k \in \mathcal{X}$ and assignment ρ can construct a deterministic finite automaton (DFA) that decides $V_{x_k}^\rho$.

Before we do this we define a DFA as $M = (Q, \Sigma, \delta, s, A)$, where Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, Σ is some alphabet, $s \in Q$ is the starting state (or source) and $A \subseteq Q$ is a set of accepting states. We use $\hat{\delta}(s, w)$ as a shorthand for $\delta(\cdots \delta(\delta(q, w_1), w_2), \ldots, w_l)$, where $(w_1, \ldots, w_l)$ are the letters of a word in Σ^*. A word $w \in \Sigma^*$ is accepted by M iff $\hat{\delta}(s, w) \in A$. We use $\hat{\delta}(w)$ as a shortcut for $\hat{\delta}(s, w)$. Further we say that a state in a DFA q is reachable from a state p, denoted $p \rightsquigarrow q$ iff there exists a word $w \in \Sigma^*$ such that $\hat{\delta}(p, w) = q$.

For any given $\mathcal{F}$ we number the regular expressions such that α_j^i is the regular expression in the jth of the match-expressions on the variable x_i, and denote by m_i the number of match-expressions on the variable x_i, hence the regular expressions that appears in $\mathcal{F}$ are: $\alpha_1^1, \ldots, \alpha_{m_1}^1, \ldots, \alpha_1^n, \ldots, \alpha_{m_n}^n$. We denote by $M_j^i = (Q_j^i, \Sigma, \delta_j^i, s_j^i, A_j^i)$ the DFAs that decides $L(\alpha_j^i)$ and hence evaluates the jth match-expression on x_i.

We define $c_j^i = \hat{\delta}_j^i(\rho(x_i))$ for all $1 \leq i \leq n, 1 \leq j \leq m_i$. Hence if the assignment ρ changes then some of the c_j^is will change. We note that $\rho \in sol(\mathcal{C})$ iff $f[match(x_i, \alpha_j^i) \leftarrow (c_j^i \in A_j^i)] = true$ for all $f \in \mathcal{F}$, where we by $f[match(x_i, \alpha_j^i) \leftarrow (q_j^i \in A_j^i)]$ denote the formula f where all match-expressions on the form $match(x_i, \alpha_j^i)$ for any i, j are replaced with the truth value of the expression $(q_j^i \in A_j^i)$. Further we note that $w^k \in V_{x_k}^\rho$ if and only if there exists some words $w^1, \ldots, w^{k-1}, w^{k+1}, \ldots, w^n \in \Sigma^*$ such that $f[match(x_i, \alpha_j^i) \leftarrow \hat{\delta}_j^i(\rho(x_i)w^i) \in A_j^i] = true$ for all $f \in \mathcal{F}$. We will assume that $M_\gamma^\beta = (Q_\gamma^\beta, \Sigma_\gamma^\beta, \delta_\gamma^\beta, s_\gamma^\beta, A_\gamma^\beta)$ for any superscript β and any or no subscript γ. Further when we mention q_γ^β we will implicitly assume that it is contained in Q_γ^β. For instance we might introduce a DFA M_j^i and refer to q_j^i without explicitly stating that $M_j^i = (Q_j^i, \Sigma_j^i, \delta_j^i, s_j^i, A_j^i)$ and that q_j^i is some state contained in Q_j^i. We will also make use of the shortcut $\gamma_1^1, \ldots, \gamma_{h_k}^k = \gamma_1^1, \ldots, \gamma_{h_1}^1, \ldots, \ldots, \gamma_1^k, \ldots, \gamma_{h_k}^k$, where $h_1, \ldots, h_k \in \mathbb{N}$.

Outline of the Paper

The goal of this paper is to construct a data structure that, based on a CSP $\mathcal{C} = (\mathcal{X}, \Sigma, \mathcal{F})$, supports three operations:

BUILD($\mathcal{C}$) that constructs a data structure based on $\mathcal{C}$ and $\rho = \{(x_1, \epsilon), \ldots, (x_n, \epsilon)\}$,

APPEND(x_k, w) that updates ρ by setting $\rho(x_k)$ to $\rho(x_k)w$ and updates the data structure accordingly.

VALIDDOMAIN(x_k) that uses the data structure to return a finite automaton that decides $V^\rho_{x_k}$.

The BUILD operation is to be run during preprocessing prior to the user interaction. Hence the goal is to make BUILD construct a data structure that supports the operations APPEND and VALIDDOMAIN as fast as possible without using too much space.

A Solution based on a single DFA

In this section we will present a construction of a DFA that for any $x_k \in \mathcal{X}$ and any assignment ρ to the variables in $\mathcal{X}$, can be turned into a finite automaton deciding $V^\rho_{x_k}$. This will prove that $V^\rho_{x_k}$ is a regular language (Theorem 1). However the data structure that will be presented in this section is too space consuming to be of any practical use.

The DFA we want to construct we denote $M_\mathcal{C}$, which is the DFA deciding a language we denote $L_\mathcal{C}$. The basic property of $L_\mathcal{C}$ is that:

$$w \in L_\mathcal{C} \iff \rho_w \in sol(\mathcal{C}) \tag{1}$$

where w is a word that *induces* the assignment ρ_w, where the meaning of induces will be defined in (2).

Intuitively we make the alphabet of $L_\mathcal{C}$, denoted $\Sigma_\mathcal{C}$, consist of all possible APPEND-operations More formally stated $\Sigma_\mathcal{C} \subset (\Sigma \cup \{\epsilon\})^n$ where each letter in $\Sigma_\mathcal{C}$ is a vector of letters from $\Sigma \cup \{\epsilon\}$ that only contains one element different from ϵ, in other words:

$$\Sigma_\mathcal{C} = \bigcup_{1 \le k \le n} \bigcup_{w \in \Sigma} \{(\underbrace{\epsilon, \ldots, \epsilon}_{k-1}, w, \underbrace{\epsilon, \ldots, \epsilon}_{n-k})\}$$

Every word w in $L_\mathcal{C}$ is a concatenation of letters from $\Sigma_\mathcal{C}$ that is $L_\mathcal{C} \subseteq \Sigma^*_\mathcal{C}$. We say that $w = w_1 \cdots w_l$ induces the assignment

$$\rho_w = \{(x_1, w_1^1 \cdots w_l^1), \ldots, (x_n, w_1^n \cdots w_l^n)\} \tag{2}$$

where w_j^i denotes the ith element in the letter $w_j \in \Sigma_\mathcal{C}$. Note that given a word $w = w_1 \cdots w_l$, we have that $\rho_{w'} = \rho_w$ for any word w' that consist of the letters $w_1, \cdots, w_l$, permuted in a way that maintains the ordering of the letters in $\{w_j \mid w_j^i \ne \epsilon\}$ for every $1 \le i \le n$.

Example 1. *Consider the CSP $\mathcal{C} = (\mathcal{X}, \Sigma, \mathcal{F})$ where $\mathcal{X} = \{x_1, x_2, x_3\}$, $\Sigma = \{a, b\}$ and $\mathcal{F}$ is some set of constraints. In this case*

$$\Sigma_\mathcal{C} = \{(a, \epsilon, \epsilon), (b, \epsilon, \epsilon), (\epsilon, a, \epsilon), (\epsilon, b, \epsilon), (\epsilon, \epsilon, a), (\epsilon, \epsilon, b)\}$$

and for instance does the word $w = (a, \epsilon, \epsilon)(\epsilon, \epsilon, a)(b, \epsilon, \epsilon)(a, \epsilon, \epsilon)$ induce the assignment $\rho_w = \{(x_1, aba), (x_2, \epsilon), (x_3, a)\}$, and so does for instance $w' = (a, \epsilon, \epsilon)(b, \epsilon, \epsilon)(a, \epsilon, \epsilon)(\epsilon, \epsilon, a)$ and $w'' = (a, \epsilon, \epsilon)(b, \epsilon, \epsilon)(\epsilon, \epsilon, a)(a, \epsilon, \epsilon)$. In the case of w, (1) becomes:

$$(a, \epsilon, \epsilon)(\epsilon, \epsilon, a)(b, \epsilon, \epsilon)(a, \epsilon, \epsilon) \in L_\mathcal{C} \iff$$

$$\{(x_1, aba), (x_2, \epsilon), (x_3, a)\} \in sol(\mathcal{C})$$

Note that for instance $w''' = (b, \epsilon, \epsilon)(a, \epsilon, \epsilon)(\epsilon, \epsilon, a)(a, \epsilon, \epsilon)$ does not induce ρ_w, since $\rho_{w'''} = \{(x_1, baa), (x_2, \epsilon), (x_3, a)\} \ne \rho_w$.

If we can construct a DFA that decides $L_\mathcal{C}$ this DFA can be used to decide for any assignment ρ whether $\rho \in sol(\mathcal{C})$. In the following we will construct such a DFA, denoted $M_\mathcal{C}$, and we will show how we based on this construction for any ρ and x_k can obtain a DFA that decides the regular language $V^\rho_{x_k}$. We construct the DFA $M_\mathcal{C}$ by setting $Q_\mathcal{C} = Q^1_1 \times \ldots \times Q^n_{m_n}$ and $\delta_\mathcal{C}((q_1^1, \ldots, q^n_{m_n}), (w^1, \ldots, w^n)) = (\delta_{\epsilon 1}^1(q_1^1, w^1), \ldots, \delta_{\epsilon m_n}^n(q^n_{m_n}, w^n))$, where the notation $\delta_{\epsilon j}^i$ is referring to δ_j^i where self-looping ϵ-transitions are added to every state in Q_j^i, that is $\delta_{\epsilon j}^i = \delta_j^i \cup \{(q_j^i, \epsilon) \to q_j^i\}$. We further set $s_\mathcal{C} = \{s_1^1, \ldots, s^n_{m_n}\}$, $c_\mathcal{C} = \hat{\delta}_\mathcal{C}(\rho(x_1), \ldots, \rho(x_n))$ and $A_\mathcal{C} = \{(q_1^1, \ldots, q^n_{m_n}) \mid f[match(x_i, \alpha_j^i) \leftarrow (q_j^i \in A_j^i)] = true$ for all $f \in \mathcal{F}\}$. We observe that many of the states in $Q_\mathcal{C}$, will be unreachable from $s_\mathcal{C}$, and therefore can be removed from the DFA.

Constructing an FA deciding $V^\rho_{x_k}$

If we want to change $M_\mathcal{C}$ such that it decides $V^\rho_{x_k}$ we set $s_\mathcal{C} = c_\mathcal{C}$ and project the alphabet on x_k – that is, we replace every letter $w = (w^1, \ldots, w^n) \in \Sigma_\mathcal{C}$ by $w^k \in \Sigma \cup \{\epsilon\}$. Note that the second step turn all transitions on w for which $w^k = \epsilon$ into ϵ-transitions, hence we have made a nondeterministic automaton on the alphabet Σ, deciding $V^\rho_{x_k}$. In Figure 1 we have illustrated the creation of $M_\mathcal{C}$ and the DFA deciding $V^\rho_{x_k}$ by an example of three match-expressions on two variables.
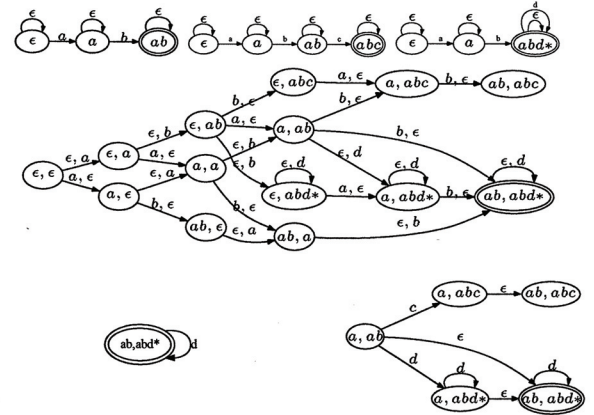

Figure 1: In this example $\mathcal{F} = \{f_1, f_2\}$ where $f_1 = match(x_2, \text{"abc"}) \lor match(x_1, \text{"ab"})$ and $f_2 = match(x_2, \text{"abd*"})$. In the top of figure we see M_1^2, M_1^1 and M_2^2, below that we see $M_\mathcal{C}$ deciding $L_\mathcal{C}$. In the bottom we see the NFA deciding $V^{\{(x_1, \text{"a"}), (x_2, \text{"ab"})\}}_{x_2}$ to the right and the corresponding DFA to the left.

The size of M_C

Though updating the values assigned to variables in $\mathcal{X}$ will be very fast using this solution, the size of M_C will be too large for the solution to be of any use for real-life problems. As an example, a problem on n variables containing a single solution $\{(x_1, w_1), \ldots, (x_n, w_n)\}$ where w_l are words of size m for all $1 \leq l \leq n$ The corresponding DFA M_C will contain $\Omega(l^n)$ states. The construction that we will achieve at the end of this paper will model the same problem using only $O(ln)$ states.

Decomposing M_C into smaller DFAs

In this section we separate the DFA M_C from last section into smaller DFAs in order to make a less space consuming construction. To be more specific we, instead of joining all the match-DFAs $M_1^1, \ldots, M_{m_n}^n$ into one large DFA as in last section, only join match-DFAs on the same variable. We construct the DFAs $M^1, \ldots, M^n$ in a way very close to the way we constructed M_C. We set $Q^i = (Q_1^i, \ldots, Q_{m_i}^i)$ and set $\delta^i((q_1^i, \ldots, q_{m_i}^i), w) = (\delta_1^i(q_1^i, w), \ldots, \delta_{m_i}^i(q_{m_i}^i, w))$ for all $1 \leq i \leq n$ and all $w \in \Sigma$. Further we set $s^i = (s_1^i, \ldots, s_{m_i}^i)$ and by setting $c^i = (c_1^i, \ldots, c_{m_i}^i)$ we get $c^i = \hat{\delta}^i(\rho(x_i))$. The alphabets in the new DFAs are Σ since following a transition in M^i corresponds to following a transition on the same letter in each of the DFAs $M_1^i, \ldots, M_{m_i}^i$. Since each position in M^i only corresponds to a subset of the positions in all $M_1^1, \ldots, M_{m_n}^n$ we are not able to encode whether $\rho \in sol(\mathcal{C})$ into the set of accepting states, as we did when we constructed M_C. This is the case because we may have assignments ρ, ρ' for which $\hat{\delta}^i(\rho(x_i)) = \hat{\delta}^i(\rho'(x_i))$ for some $1 \leq i \leq n$ even though $\rho \in sol(\mathcal{C}) \wedge \rho' \notin sol(\mathcal{C})$ caused by some difference in the values assigned to $\mathcal{X} \setminus \{x_i\}$ by ρ and ρ' respectively. As an alternative to the concept of the *set of accepting states* we introduce the concept of an *acceptance value*, that will be used to compute what corresponds to A_C when the DFAs deciding valid domains have to be constructed during the interactive configuration. We denote the acceptance value of M^i for any $1 \leq i \leq n$ by a^i and define it by $a^i(q^i) = (q_1^i \in A_1^i, \ldots, q_{m_i}^i \in A_{m_i}^i)$

By specifying a^i instead of A^i we have deviated from the definition of a DFA. We therefore define a *Multi-DFA* (MDFA) by: $M = (Q, \Sigma, \delta, s, a)$ as a DFA that has no set of accepting states A, but instead has a function $a : Q \to \mathbb{B}^k$. Observe that if $k = 1$ the MDFA can be translated into a standard DFA by replacing the function a by the set $A = \{q \mid a(q) = true\}$.

Constructing an MDFA

We might build an MDFA by slightly modifying the construction of the DFA M_C. However this might make the intermediate structure very large. Instead we use a simple approach making a simultaneous DFS in the DFAs that has to be joined as described in the pseudocode below. We let $\mu, Q^k, \delta, s^k, a^k, m_k$ and $M_j^k = (Q_j^k, \delta_j^k, s_j^k, a_j^k)$, for $1 \leq j \leq m_k$ be globals.

$\text{REC}(q_1^k, \ldots, q_{m_k}^k)$
1 **if** $\mu(q_1^k, \ldots, q_{m_k}^k)$ is defined
2   **then return** $\mu(q_1^k, \ldots, q_{m_k}^k)$
3 create a new state $q^k \notin Q^k$
4 $Q^k \leftarrow Q^k \cup \{q^k\}$
5 $\mu(q_1^k, \ldots, q_{m_k}^k) \leftarrow q^k$
6 $a^k(q^k) \leftarrow \big((q_1^k \in A_1^k), \ldots, (q_{m_k}^k \in A_{m_k}^k)\big)$
7 **for** each $w \in \Sigma$
8   **do** $\delta^k(q^k, w) \leftarrow \text{REC}(\delta_1^k(q_1^k, w), \ldots, \delta_{m_k}^k(q_{m_k}^k, w))$
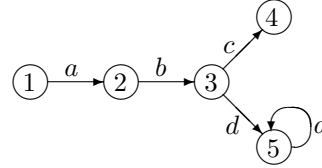9 **return** q^k

$\text{CONSTRUCTMDFA}(M_1^k, \ldots, M_{m_k}^k)$
1 $Q^k \leftarrow \delta^k \leftarrow a^k \leftarrow \mu \leftarrow \emptyset$
2 $s^k \leftarrow \text{REC}(s_1^k, \ldots, s_{m_k}^k)$
3 **return** $(Q^k, \delta^k, s^k, a^k)$

For instance the constraint $match(x_1, \text{"abc"}) \wedge match(x_1, \text{"abd} * \text{"})$ on x_1 will result in the MDFA drawn in Figure 2.

Acceptance values
$1 : (false, false)$
$2 : (false, false)$
$3 : (false, true)$
$4 : (true, false)$
$5 : (false, true)$

Figure 2: The MDFA deciding $L(\text{"abc"})$ and $L(\text{"abd}*\text{"})$

By defining minimization by the straight forward generalization as minimization of DFAs, we can prove a further property of the algorithm CONSTRUCTMDFA.

Definition 2. *A MDFA is minimized if all states in the MDFA are reachable from the source and no pair of states in the MDFA are equivalent. In an MDFA M, any pair of nodes $p, q \in Q : p$ and q are equivalent iff for all words $w \in \Sigma^*$: $a(\hat{\delta}(p, w)) = a(\hat{\delta}(q, w)))$.*

Lemma 1. *If the DFAs given as input to* CONSTRUCT-MDFA *are minimized then the constructed MDFA will be minimized.*

Proof. We first note that all states in Q^k are reachable from s^k. This is due to the fact that every state created except s^k will be a result of a recursive call made at line 8. Hence every created state in the MDFA will be assigned to a $\delta^k(q^k, w)$, where $s^k \rightsquigarrow q^k$ and $w \in \Sigma$.

We then prove that no pair of states in the constructed MDFA M^k is equivalent if the DFAs $M_1, \ldots, M_{m_k}$ is minimal. Consider any pair of distinct nodes $p^k, q^k \in Q^k$. Suppose $\mu(p_1^k, \ldots, p_{m_k}^k) = p^k$ and $\mu(q_1^k, \ldots, q_{m_k}^k) = q^k$. Since $p^k \neq q^k$ we know by the initial check on line 1-2 that $(p_1^k, \ldots, p_{m_k}^k) \neq (q_1^k, \ldots, q_{m_k}^k)$. Hence for some $1 \leq j \leq m_k$ we have $p_j^k, q_j^k \in Q_j^k$ for which $p_j^k \neq q_j^k$. Since M_j^k is minimized we know that p_j^k is not equivalent to q_j^k which implies that there exists a $w \in \Sigma^*$ for

which $a(\hat{\delta}_j^k(p_j^k, w)) \neq a(\hat{\delta}_j^k(q_j^k, w))$. This implies that $a^k(\hat{\delta}^k(\mu(p_1^k, \ldots, p_{m_k}^k), w)) \neq a^k(\hat{\delta}^k(\mu(q_1^k, \ldots, q_{m_k}^k), w))$ which is the same as $a^k(\hat{\delta}^k(p^k, w)) \neq a^k(\hat{\delta}^k(q^k, w))$. Hence p^k and q^k are not equivalent. $\square$

Constructing a DFA deciding $V_{x_k}^\rho$

Suppose for some MDFAs $M^1, \ldots, M^n$ on $\mathcal{F}$ and some assignment ρ of values to the variables in $\mathcal{X}$, that we want to construct the DFA $M' = (Q', \Sigma', \delta', s', A')$ deciding $V_{x_k}^\rho$ for some x_k. To attain this we first set $s' = c^k$ and $Q' = \{q \in Q^k \mid s \rightsquigarrow q\}$. Further we set $\delta' = \delta^k$ and $\Sigma' = \Sigma$. We now only need to compute the set of accepting states A'. We will devote the rest of this section to construct an auxiliary data structure that will provide an efficient computation of A'.

Let $Y = y_1^1, \ldots y_{m_n}^n$ be a set of boolean variables, and define $y^i = (y_1^i, \ldots, y_{m_i}^i)$ for $1 \leq i \leq n$. We want to set some constraints on the variables in Y that are satisfiable iff $\forall_{1 \leq i \leq n} : y^i = a^i(\hat{\delta}^i(\rho\rho'(x_i)))$ for some ρ' for which $\rho\rho' \in sol(\mathcal{C})$ where ρ is the current assignment to $\mathcal{X}$ in the interactive configuration. This can be attained by two constraints: We first say the value of y^i has to correspond to some acceptance value of some state in M^i that is reachable from the source of M' that is:

$$g_\rho = \bigwedge_{1 \leq i \leq n} \bigvee_{c^i \rightsquigarrow q^i} y^i = a^i(q^i) \quad (3)$$

Further evaluating the match-expressions by the y-values has to make $\mathcal{F}$ true, that is:

$$g_\mathcal{F} = \bigwedge_{f \in \mathcal{F}} f[match(x_i, \alpha_j^i) \leftarrow y_j^i] \quad (4)$$

Using these two constraints we can attain for any $w \in \Sigma^*$ that $w \in V_{x_k}^\rho$ iff $g_\rho \wedge g_\mathcal{F} \wedge (y^k = a^k(\rho(x_k)w))$ is satisfiable. Hence we get for the DFA deciding $V_{x_k}^\rho$ that:

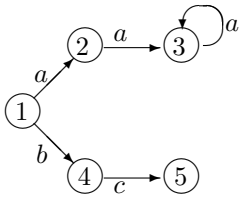$$A' = \{q^k \in Q^k \mid g_\rho \wedge g_\mathcal{F} \wedge y^k = a^k(q^k) \text{ is satisfiable}\} \quad (5)$$

Acceptance values
$1 : (false, false)$
$2 : (true, false)$
$3 : (false, true)$
$4 : (false, false)$
$5 : (false, true)$

Figure 3: MDFA deciding $L(\text{``}a\text{''})$ and $L(\text{``}(a*)|(bc)\text{''})$

Example 2. *Suppose for the CSP $\mathcal{C} = (\mathcal{X}, \Sigma, \mathcal{F})$ that $\mathcal{X} = \{x_1, x_2\}$ and $\mathcal{F} = \{f_1, f_2\}$ where $f_1 = match(x_1, \text{``}a\text{''}) \vee match(x_2, \text{``}def\text{''})$ and $f_2 = match(x_1, \text{``}(a*)|(bc)\text{''})$. The MDFA evaluating the match-expressions on x_1 is showed in Figure 3.*

Since we initially want the compute the DFAs deciding the valid domains of the variables x_1 and x_2 given the assignment $\rho = \{(x_1, \epsilon), (x_2, \epsilon)\}$, we will have to use (5) in order to compute the set of accepting states A' for each of these DFAs. In order to use (5) we have to compute the set of assignments to the Y-variables that satisfy $g_\rho \wedge g_\mathcal{F}$.

We start out by computing the constraints g_ρ and $g_\mathcal{F}$. By (3) we get $g_\rho = (y^1 = (false, false) \vee y^1 = (false, true) \vee y^1 = (true, false)) \wedge (y^2 = (false) \wedge y^2 = (true))$. By (4) we get $g_\mathcal{F} = (y_1^1 \vee y_1^2) \wedge y_2^1$.

We now impose the constraints $g_\rho \wedge g_\mathcal{F}$ on the values of the variables in Y. By g_ρ, we get that $y^1 \neq (true, true)$ and by $g_\mathcal{F}$ we get that $y_2^1 = true$. By $y^1 = (y_1^1, y_2^1) \neq (true, true)$ and $y_2^1 = true$ we imply that $y^1 = (false, true)$. Since we now have $y_1^1 = false$ the value assigned to y_1^2 has to be true in order for $g_\mathcal{F}$ to be satisfied. Therefore the only assignment to the variables in Y that satisfies $g_\rho \wedge g_\mathcal{F}$ is $\{(y_1^1, false), (y_1^2, true), (y_2^1, true)\}$. If we construct the DFA that decides $V_{x_1}^{\{(x_1, \epsilon), (x_2, \epsilon)\}}$ then only the states numbered 3 and 5 n Figure 3 are accepting states according to (5), hence $V_{x_1}^{\{(x_1, \epsilon), (x_2, \epsilon)\}} = \{\text{``}a*\text{''}, \text{``}bc\text{''}\}$. We can also infer that $V_{x_2}^{\{(x_1, \epsilon), (x_2, \epsilon)\}} = \{\text{``}def\text{''}\}$

Computing reachable acceptance values

In order to prevent that we have to visit all states reachable from c^i in every M^i each time we have to check g_ρ for satisfiability we will introduce the notion of the *set of reachable acceptance values*. We define the set of reachable acceptance values by

$$R^i(p^i) = \{a^i(q^i) \mid p^i \rightsquigarrow q^i\}$$

and we define the notation $y^i \in B^i$ as $\bigvee_{b^i \in B_i} y^i = b^i$. Using this we can rewrite g_ρ as $g_\rho = \bigwedge_{1 \leq i \leq n} y^i \in R^i(c^i)$ Instead of recomputing the constraint g_ρ every time ρ (and thereby some of the $c^1, \ldots, c^n$) changes we will compute $R^i(q^i)$ for every $q^i \in Q^i$ and all $1 \leq i \leq n$ during preprocessing.

In the computation of the reachable acceptance values we use the fact that every strongly connected component will have the same set of reachable acceptance values. A strongly connected component (SCC) in an (M)DFA M we define as a set of states $C \subseteq Q$ for which it for any $p \in C$ holds that $p \rightsquigarrow q$ and $q \rightsquigarrow p$ iff $q \in C$. Calculating the SCCs in an MDFA be done by making two DFS in the MDFA (Cormen *et al.* 2001). In the following algorithm we assume that M is an MDFA:

REACHABLEACCEPTANCEVALUES(M)

```
1   let C' be the set SCCs in Q
2   for each C_1, C_2 ∈ C'
3       do if δ(p, w) = q for some p ∈ C_1, q ∈ C_2, w ∈ Σ
4           then Γ(C_1) ← Γ(C_1) ∪ {C_2}
5   for each C ∈ C'
6       do R'(C) ← {a(q) | q ∈ C}
7   for each C_1 ∈ C' in reverse topological order
8       do R'(C_1) ← R'(C_1) ∪ {R'(C_2) | C_2 ∈ Γ(C_1)}
9   for each C ∈ C'
10      do for each q ∈ C
11          do R(q) ← R'(C)
12  return R
```

The Algorithms in Pseudocode

In this section we will present the three algorithms BUILD($\mathcal{C}$), APPEND(x_k, w) and VALIDDOMAIN(x_k). The first algorithm BUILD constructs the data structure the second algorithm APPEND updates the data structure when ρ is updated and the third algorithm VALIDDOMAIN(x_k) returns a DFA deciding $V_{x_k}^\rho$.

In all algorithms we assume that $g, M_1, \ldots, M_n, R^1, \ldots, R^n, a^1, \ldots, a^n, c^1, \ldots, c^n, \Sigma$ and ρ are globals. We further assume that initially $\rho \leftarrow \{(x_1, \epsilon), \ldots, (x_n, \epsilon)\}$ and $m_1, \ldots, m_n = 0$

BUILD($\mathcal{C}$)

1   **for** $i \leftarrow 1$ **to** n
2   **do for** each $match(x_i, \alpha_j^i)$ occurring in $\mathcal{F}$
3   **do** $m_i \leftarrow m_i + 1$
4   Build a DFA M_j^i on $L(\alpha_j^i)$
5     $y^i \leftarrow (y_1^i, \ldots, y_{m_i}^i)$
6     $M^i \leftarrow$ CONSTRUCTMDFA($M_1^i, \ldots, M_{m_i}^i$)
7     $c^i \leftarrow s^i$
8     $R^i \leftarrow$ REACHABLEACCEPTANCEVALUES(M^i)
9   $g_\rho = \bigwedge_{1 \leq i \leq n} y^i \in R^i(c^i)$
10 $g_{\mathcal{F}} = \bigwedge_{f \in \mathcal{F}} f[match(x_i, \alpha_j^i) \leftarrow y_j^i]$
11 $g \leftarrow g_\rho \wedge g_{\mathcal{F}}$
12 **if** g is unsatisfiable
13   **then error** "No feasible solutions to $\mathcal{C}$"

Line 1-8 constructs the MDFAs on the match-expressions in $\mathcal{F}$. Line 9-11 computes g as $g_{\mathcal{F}} \wedge g_\rho$. Line 12-13 checks whether $sol(\mathcal{C}) \neq \emptyset$ by checking whether g is satisfiable.

VALIDDOMAIN(x_k)

1 $A' \leftarrow \emptyset$
2 **for** each $q^k \in Q^k$
3   **do if** $g \wedge (y^k = a^k(q^k))$ is satisfiable
4   **then** $A' \leftarrow A' \cup \{q^k\}$
5 $M' \leftarrow (Q^k, \Sigma, \delta^k, c^k, A')$
6 **return** M'

This algorithm construct a DFA deciding $V_{x_k}^\rho$ based on the MDFA on x_k. The set of accepting states A' in Line 2-4 is done by using (5).

APPEND(x_k, w)

1 **if** $g \wedge (y^k \in R^k(\delta^k(c^k, w)))$ is unsatisfiable
2   **then error** "invalid append to variable in $\mathcal{X}$"
3 $\rho(x_k) \leftarrow \rho(x_k)w$
4 $c^k \leftarrow \delta^k(c^k, w)$
5 $Q^k \leftarrow \{q^k \in Q^k \mid s^k \rightsquigarrow q^k\}$
6 $g \leftarrow g \wedge (y^k \in R^k(c^k))$

The algorithm APPEND updates ρ by appending w to the value assigned to x_k by ρ. This update entails that c^k has to be updated and thereby Q^k can be pruned by removing states that are unreachable by the new c^k. We update g to conform with the new c^k by setting $g \leftarrow g \wedge (y^k \in R^k(c^k))$.

Implementation

In the pseudocode from last section the main issue is to decide whether g is satisfiable in Line 3 in VALIDDOMAIN. This can be done by using the DPLL-algorithm (Davis, Logemann, & Loveland 1962), however in the setting of interactive configuration, where fast decisions on whether g is satisfiable have to be available, encoding g as a BDD seems to be the obvious choice as we can decide whether g is satisfiable in linear time in the size of the BDD-representation of g by using a specialization of the valid domains computation mentioned in (Tarik Hadzic 2006).

In standard interactive configuration as in (Hadzic *et al.* 2004) where the variables, say $\mathcal{Z}$, have small finite domains and a constraint, say e, are compiled into the BDD G_e, prior to the interactive configuration, the only way that G_e is updated during the interactive configuration is by restriction, that is by assigning values to variables in $\mathcal{Z}$. Since applying a restriction on a BDD takes time linear in the size of the BDD and since a restriction for the purpose of valid domains computation can be done without increasing the size of the BDD, it is known at compile time that, during the entire interactive configuration, the size of G_e and the running time of each restriction on G_e will be linear in the size of the initial BDD-representation of e that was compiled prior to the interactive configuration.

In our approach things are different since we need to set $G_g \leftarrow G_g \wedge G_{y^i \in R^i(c^i)}$ during the interactive configuration. The size of the BDD-representation of $G_1 \wedge G_2$ is $O(|G_1||G_2|)$ in the worst case (Bryant 1986). However since the constraints $y^1 \in R^1(q^1), \ldots, y^n \in R^n(q^n)$ share no variables the size of $G_{\bigwedge_{1 \leq i \leq n} y^i \in R^i(c^i)}$ is bounded by $O\big(\sum_{1 \leq i \leq n} max\{|G_{q^i \in R^i(q^i)}| \mid c^i \rightsquigarrow q^i\}\big)$, hence we can bound the size $G_{g_\rho} = \bigwedge_{1 \leq i \leq n} G_{y^i \in R^i(c^i)}$ by the sum of largest BDD representation of the reachable acceptance values in each of MDFAs. To bound the size of $G_g = G_{g_{\mathcal{F}}} \wedge G_{g_\rho}$ we take the product of the size of G_{g_ρ} and the size of the BDD representation of $\mathcal{F}$, hence we get:

$$|G_g| = O\big(|G_{\mathcal{F}}| \sum_{1 \leq i \leq n} max\big\{|G_{\{y^i \in R^i(q^i)\}}| \mid q^i \in Q^i\big\}\big)$$

The bound on the size of G_g is a very pessimistic one, in most practical cases the result of computing the and of two BDDs will be much smaller than stated by the bound.

The asymptotic running time of APPEND is dominated by the update of g in Line 6. We can therefore bound the asymptotic running time of APPEND(x_k, w) by:

$$O(|G_{\{y^k \in R^k(s^k)\}}||G_g|)$$

The time complexity of VALIDDOMAIN(x_k) is dominated by the time, in Line 3, used to check for each state whether it is should be an accepting state in the DFA that is whether or not $g \wedge (y^k = a^k(q^k))$ is satisfiable. We note that the asymptotic time complexity of deciding whether $g \wedge (y^k = a^k(q^k))$ is satisfiable is linear in the number of variables in Y. If we use dynamic programming on the computation of $a^k(q^k)$ we only need to compute $G_g \wedge G_{y^k = a^k(q^k)}$ once for each different acceptance value. Hence the asymptotic worst-case

running time of VALIDDOMAIN(x_k) is:

$$O(|Y||a^k| + |Q^k|)$$

where $|a^k| = |\{a^k(q^k) \mid q^k \in Q^k\}|$ and Q^k is the set of states reachable from s^k in the MDFA on x_k.

As a guarantee on the space usage of the data structure we have the size of G_g, the size of each of the MDFAs, and the size of the BDD-representation of all different sets of reachable accepting states, that is:

$$O\Big(|G_g| + \sum_{1 \leq i \leq n} |M^i| + \sum_{R \in R'} G_R\Big)$$

where $O(|M^i|)$ is the set of transitions in M^i and $R' = \{R^i(q^i) \mid q^i \in Q^i \wedge 1 \leq i \leq n\}$.

Combining regular and finite domains

Roughly stated we can encode a set of constraints $\mathcal{H}$ on a set of integer variables $\mathcal{Z}$ with discrete finite domains by adding the $\mathcal{Z}$ to Y and adding $\mathcal{H}$ to $g_{\mathcal{F}}$. For instance the constraint country = "Denmark" $\implies$ vat = 25, where country is a string variable and vat is an integer on the domain $\{0, \ldots, 100\}$, will be encoded as $\neg match(x_1, \text{"Denmark"}) \vee x_2 = 25$, hence we get $Y = \{y_1^1, y_1^2, \ldots, y_7^2\}$ and $g_{\mathcal{F}} = \neg y^1 = (1) \vee y^2 = (0, 0, 1, 1, 0, 0, 1)$ where 0011001 is the binary encoding of 25. If e.g. the user chooses $vat = 10$ the constraint $g_\rho = (y^2 = (0, 0, 0, 1, 0, 1, 0))$ is added to g which implies $y^1 = false$ and thereby $L(\text{"Denmark"}) \nsubseteq V_{x_1}^\rho$. Note that in this case g_ρ is a restriction. Hence if all variables are integer variables we have an instance of a standard interactive configuration problem as described in (Hadzic *et al.* 2004), and the computations of valid domains will be computed in exactly the same way.

Conclusion

In this paper we have introduced a data structure that is able to handle interactive configuration on string-variables by using a combination of regular expressions and boolean logic. We have shown that this can be modeled by a single large DFA and argued that the size of this DFA is too large be of any practical use. Further we have made a data structure that simulates the large DFA by a combination of DFAs (or Multi-DFAs to be precise) and a BDD.

We consider the work in this paper as the first step in a new path in the area of interactive configuration. In this first step the primary goal has been to provide a general and solid framework that will serve as the starting point for future research.

Acknowledgement

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs-Application to configuration . *Artificial Intelligence* 135(1-2):199–234. bb modif 28/02/02.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Commun. ACM* 5(7):394–397.

Golden, K., and Pang, W. 2003. Constraint reasoning over strings. In *CP*, 377–391.

Hadzic, T.; Subbarayan, S.; Jensen, R. M.; Andersen, H. R.; Hulgaard, H.; and Møller, J. 2004. Fast backtrack-free product configuration using a precompiled solution space representation. In *Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems*, 131–138. DTU-tryk.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, 482–495. Springer.

Raskin, J. 2000. *The Humane Interface*. Addison Wesley.

Tarik Hadzic, Rune Møller Jensen, H. R. A. 2006. Calculating valid domains for bdd-based interactive configuration. arXiv.