Formalizing Configuration Knowledge Using Rules with Choices

Timo Soininen
Helsinki University of Technology
TAI Research Center and
Lab. of Information Processing Science
P.O.Box 9555, FIN-02015 HUT, Finland
Timo.Soininen@hut.fi

Ilkka Niemelä
Helsinki University of Technology
Dept. of Computer Science and Eng.
Digital Systems Laboratory
P.O.Box 1100, FIN-02015 HUT, Finland
Ilkka.Niemela@hut.fi

Abstract

This paper studies the formal underpinnings of the product configuration problem. A rule-based language is proposed for expressing typical forms of knowledge interactions in configuration models, i.e. choices, dependencies between choices, optionality and defaults. The language is equipped with a declarative semantics that provides formal definitions for the main concepts in product configuration, i.e., configuration model, customer requirements, satisfiability of requirements and validity of a configuration. A key feature is that the semantics ensures that valid configurations are tightly grounded in the configuration rules by employing Horn clause derivability but without resorting to an explicit minimality condition on configurations. This type of groundedness has not been considered in previous work on product configuration. Avoiding minimality conditions has a favorable effect on the complexity of the configuration tasks. For example, the validity of a configuration as well as satisfiability of requirements can be decided in polynomial time and also other computational task remain in **NP**. The semantics of the rule-based language is shown to be closely related to the stable model semantics of normal logic programs and the possible model semantics of disjunctive programs. It is argued that from a knowledge representation point of view the rule-based language is more attractive for representing configuration knowledge than propositional logic or constraints.

1 [1]Introduction

Product configuration has been a fruitful topic of research in artificial intelligence for the past two decades (see, e.g. [McD82, MF89, CGS+89, HJ91, Kle96]). In the last five years product configuration has also become a commercially successful and important application of artificial intelligence techniques. Knowledge-based systems (KBS) employing techniques such as constraint satisfaction [Tsa93] have been successfully applied to product configuration. The reason for the success of KBSs in this area is that the relevant knowledge is usually well-defined and complete [Kle96]. However, there is no formal model of product configuration that covers all the aspects of the problem. Most of the work in the field has been based on informal approaches to knowledge representation and reasoning with few notable exceptions (e.g. [MF90, HS93, Kle96]). This makes it difficult to compare the different approaches, analyze the computational complexity of the related decision problems and to develop efficient implementation techniques.

In this paper we present a formal model of typical interactions in configuration knowledge. For formalizing the interactions, techniques developed in nonmonotonic reasoning research turn out to be relevant. We develop a rule language for representing the configuration knowledge and propose a semantics for it. This formalization differs from most research on product configuration in that a configuration must be *grounded*, i.e. everything in a configuration must have a justification. This could be achieved by requiring that a configuration is minimal. However, for practically relevant product configuration problems the basic decision problem of deciding if a configuration is valid should be tractable. Minimization-based approaches often lead to higher complexity. Instead, we apply derivability from a set of Horn clauses as a weaker form of groundedness. We analyze the complexity of main decisions problems for the language and show that tractability of the basic decision problem is preserved. The language is argued to be more attractive than propositional logic or constraints for representing configuration knowledge.

In Section 2 we define the configuration problem and discuss typical knowledge interactions. We define a rule language for representing the knowledge and a non-monotonic semantics for it in Section 3 and give several examples. In Section 4 we discuss the relations of the proposed semantics to other non-monotonic semantics. In Section 5 the computational complexity of the main decision problems for the language is discussed and expressiveness of the language is compared to that of propositional logic. We compare our formalization with previous work on product configuration in Section 6. In Section 7 we present conclusions and discuss future work in this area.

2 Product Configuration Domain

Product configuration is roughly defined as the problem of producing a specification of a product individual as a collection of predefined components. The inputs of the problem are a *configuration model*, which describes the components that can be included in the configuration and the rules on how they can be combined to form a working product, and *customer requirements* that specify some properties that the product individual should have. The output is a *configuration*, an accurate enough description of a product individual to be manufactured. The configuration must *satisfy* the customer requirements and be *valid* in the sense that it does not break any of the rules in the configuration model and it consists only of the components grounded in the configuration model.

This definition of product configuration does not adequately capture all aspects of configuration problems. Missing features include representing and reasoning about attributes, structure and connections of components, resource production and use by components [MF89, CGS+89, HJ91] and optimality of a configuration. Our definition is a simplification for the purpose of an abstract level analysis. It is intended as the foundation on which further aspects of product configuration can be defined. Correspondingly, we use the term *element* to mean any relevant piece of information on a configuration. An element can, for example, be a component or it may represent other aspects.

A *product configurator* is a KBS that is capable of representing the knowledge included in configuration models, requirements and configurations. In addition, it is capable of (i) *checking* whether a configuration is valid with respect to the configuration model and satisfies a set of requirements and/or (ii) *generating* one or all valid configuration(s) for a configuration model and a set of requirements.

As an example of a configurable product, consider a PC. The components that occur in the configuration model of a PC include different types of display units, hard disks, CD-ROM drives, floppy drives, extension cards and so on. These have certain rules on how they can be combined with each other to form a working product. For example, in a typical configuration model a PC would be defined to have a mass-storage which must be chosen from a set of alternatives, e.g. an IDE-hard-disk, SCSI-hard-disk and a floppy drive. A computer would also need a keyboard, which could have either a Finnish or United Kingdoms layout. Having a SCSI-hard disk in the configuration of a PC would typically require that an additional SCSI-controller would be included in the configuration as well. In addition, a PC may optionally have a CD-ROM drive. A configuration model for a PC might also define that unless otherwise specified, an IDE-hard-disk will be the default choice for mass-storage.

The fundamental form of knowledge in a configuration model is that of a *choice*. There are basically two types of choices. Either at least one or exactly one of alternative elements must be chosen. Whether a choice must be made may depend on some set of elements being in the configuration. Other forms of configuration knowledge include the following:

- An element is *optional*. Optional elements can be chosen into a configuration or they can be left out.
- A set of elements in the configuration *requires* some set of elements to be in the configuration as well [TSMS96, Kle96].
- A set of elements are *incompatible* with each other [TSMS96, Kle96].
- An element is a *default*. It is in the configuration unless something else has been specified.

The last three types of knowledge define how elements resulting from choices can be combined.

In the following section we introduce a language for representing configuration knowledge. For simplicity, we have kept the number of primitives in the language low. However, it has been widely recognized in configuration research that understandability and maintainability are key factors in the success of using a product configurator. In a strictly minimal language some knowledge would have to be encoded through rather artificial constructions that are difficult to understand and maintain. In the next section we give several examples on how other relevant constructs can be seen as short-hand notation.

3 Configuration Rule Language

The configuration rule language **CRL** introduced in this section captures typical knowledge interactions in the configuration domain at an abstract level. The idea is to focus on the interactions of elements and not on the primitives of a particular configuration knowledge modeling language.

The basic construction blocks of the language are primitive propositional atoms, which are combined through a set of connectives into rules. We assume for simplicity that primitive propositional atoms can be used to represent elements. We define a *configuration model* as a set of **CRL** rules and *customer requirements* as another set of rules. A *configuration* is defined as a set of atoms.

The syntax of **CRL** is defined as follows. The alphabet of **CRL** consists of the connectives ",", "$\leftarrow$", "$|$", "not", parentheses and atomic propositions. The connectives are read as "and", "requires", "or" and "not", respectively. The rules in **CRL** are of the form

$$a_1 \mid \cdots \mid a_l \leftarrow b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n)$$

where $a_1, \ldots, a_l$, $b_1, \ldots, b_m$ and $c_1, \ldots, c_n$ are atoms, $l \geq 0$, $m \geq 0$, $n \geq 0$. For convenience, we refer to the subset of a set of rules R with exactly one atom in the head as *requires-rules*, R_{r}, rules with more than one atom in the head as *choice-rules*, R_{c}, and rules with no atoms in the head as *incompatibility-rules*, R_{i}. In the definitions we treat requires-rules as a special case of choice-rules with only one alternative in the head.

Example 1 A very simple configuration model R_{PC} of the PC-example above (without the optional CD-ROM and default mass-storage) could consist of the following rules:

$$\begin{array}{l} computer \leftarrow \\ IDEdisk \mid SCSIdisk \mid floppydrive \leftarrow computer \\ FinnishlayoutKB \mid UKlayoutKB \leftarrow computer \\ \leftarrow FinnishlayoutKB, UKlayoutKB \\ SCSIcontroller \leftarrow SCSIdisk \end{array}$$

■

Next we define when a configuration satisfies a set of rules and is valid with respect to a set of rules. We say that a configuration *satisfies customer requirements* if it satisfies the corresponding set of rules.

Definition 3.1 *A configuration C* satisfies *a set of rules R in* **CRL**, *denoted by $C \models R$, iff the following two conditions hold:*

(i) If $a_1 \mid \cdots \mid a_l \leftarrow b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n) \in R_{\mathrm{r}} \cup R_{\mathrm{c}}$, $\{b_1, \ldots, b_m\} \subseteq C$, and $\{c_1, \ldots c_n\} \cap C = \emptyset$, then there exists an i such that $1 \leq i \leq l$ and $a_i \in C$.

(ii) If $\leftarrow b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n) \in R_i$, then at least one of the following is true: there exists an i such that $1 \leq i \leq m$ and $b_i \notin C$, or, $\{c_1, \ldots, c_n\} \cap C \neq \emptyset$.

In order to define the validity of a configuration, we define an operator $\Gamma_C(R)$ that is a transformation of a set of rules R in **CRL**.

Definition 3.2 *Given a configuration C and a set of rules R in* **CRL**, *we denote by $\Gamma_C(R)$ the set of rules*

$$\begin{array}{l} \{a_i \leftarrow b_1, \ldots, b_m : \\ a_1 \mid \cdots \mid a_l \leftarrow b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n) \in R, \\ a_i \in C, 1 \leq i \leq l, \{c_1, \ldots c_n\} \cap C = \emptyset\} \end{array}$$

The result of the transformation is a set of Horn clauses if we interpret the symbols "$\leftarrow$" and "," as classical implication and conjunction, respectively. Under this interpretation the reduct has a unique least model.We denote the least model of a set R of Horn clauses by $\mathrm{MM}(R)$. Notice that the least model coincides with the set of atoms logically entailed by the clauses and also with the set of atoms derivable by interpreting the clauses as inference rules.

The intuition behind the transformation is that, given a choice rule, if any of the alternatives in the head of the rule are chosen, then the reduct of the transformation includes a rule that can justify the choice (if the body of the rule can be justified). If some alternative is not chosen, then there is no need for the choice to be justified and consequently no corresponding rules are included. The default negation "not($\cdot$)" is handled using a technique similar to that in the stable model semantics of logic programs [GL88].

Definition 3.3 *Given a configuration C and a set of rules R in* **CRL**, *C is R-valid iff $C = \mathrm{MM}(\Gamma_C(R))$ and $C \models R$.*

The intuition behind this definition is as follows: the first fix-point condition guarantees that a configuration must be grounded in the rules. All the things in the configuration are derivable from (the reduct of) the configuration rules. On the other hand, everything that can be derived using (the reduct of) the rules must be in the configuration. The second condition ensures that all the necessary choices have been made and all the requires- and incompatibility-rules are respected.

Example 2 Consider the configuration model R_{PC} in Example 1 and the simple set of customer requirements $\{FinnishlayoutKB \leftarrow\}$. The configuration

$$C_1 = \{computer, SCSIdisk, UKlayoutKB\}$$

does not satisfy the configuration model nor the customer requirements according to Definition 3.1 and thus it is not R_{PC}-valid, either. The configuration

$$C_2 = \{computer, IDEdisk, FinnishlayoutKB, SCSIcontroller\}$$

does satisfy the configuration model and the customer requirements. However, it is not R_{PC}-valid because the reduct $\Gamma_{C_2}(R_{PC})$ is

$$\begin{array}{l} computer \leftarrow \\ IDEdisk \leftarrow computer \\ FinnishlayoutKB \leftarrow computer \\ SCSIcontroller \leftarrow SCSIdisk \end{array}$$

The minimal model $\mathrm{MM}(\Gamma_{C_2}(R_{PC}))$ of the reduct does not contain the *SCSIcontroller* and thus it is not equal to C_2. Consider now the configuration

$$C_3 = \{computer, SCSIdisk, FinnishlayoutKB, SCSIcontroller\}$$

This configuration is R_{PC}-valid and satisfies the customer requirements. ∎

We now present some examples of typical configuration knowledge formulated as **CRL** rules and the corresponding configurations. These examples give an idea of how to represent *exclusive choice*, *optional choice* and *default* alternatives in **CRL**.

Example 3 Consider first the set of rules R_1 in Figure 1. The valid configurations with respect to R_1 are $\{c, a\}$, $\{c, b\}$ and $\{c, a, b\}$. The reducts of R_1 with respect to these configurations are $\{a \leftarrow c; c \leftarrow\}$,

R_1 :	R_2 :	R_3 :
$a \mid b \leftarrow c$	$a \mid b \leftarrow$	$a \mid b \leftarrow c$
$c \leftarrow$	$\leftarrow a, b$	$c \mid c' \leftarrow d$
		$\leftarrow c, c'$
		$d \leftarrow$

Figure 1: Examples of rules

$\{b \leftarrow c; c \leftarrow\}$ and $\{a \leftarrow c; b \leftarrow c; c \leftarrow\}$, respectively. It is easy to see that the minimal models of these reducts are exactly the configurations and that the configurations satisfy the rules in R_1. Now, consider the case where the latter rule is omitted. In this case, the only valid configuration is the empty configuration $\{\}$, since a and b cannot have a justification.

The rule set R_2 encodes an exclusive choice. The valid configurations with respect to R_2 are $\{a\}$ and $\{b\}$. One must choose one of $\{a, b\}$ to any configuration, and choosing either a or b prohibits the other from being chosen as they are incompatible.

The first three rules in R_3 demonstrate how to represent an optional choice rule whose head consists of the atoms a and b and whose body is d. The valid configurations with respect to R_3 are $\{c', d\}$, $\{a, c, d\}$, $\{b, c, d\}$ and $\{a, b, c, d\}$. In this example either c or c' must be in a configuration. These additional atoms represent the cases where the choice is made and not made, respectively. Now, consider the rule set obtained by adding the rule $a \leftarrow \mathrm{not}(b), d$ to R_3. The valid configurations are now $\{c', a, d\}$, $\{c, a, d\}$, $\{c, b, d\}$ and $\{c, a, b, d\}$. This rule set represents a default choice (a is the default) which is made unless one of the alternatives is explicitly chosen. ∎

4 Relationship to Nonmonotonic Semantics

The configuration rule language **CRL** resembles disjunctive logic programs and deductive databases. The semantics is also similar to logic programming semantics. As we pointed out, the default negation is handled using a construct closely related to the stable model semantics of logic programs [GL88]. In fact, if we limit the language to requires-rules, normal logic programs with the stable model semantics is obtained: for a set of requires-rules, a set of atoms is a valid configuration iff it is a stable model of the set of rules.

When choice-rules are allowed, our semantics differs from the usual semantics of disjunctive logic programs such as the disjunctive stable model semantics [GL91], D-WFS [BD95], and STATIC [Prz95]. The main difference is that these semantics have subset minimality of the models as a built-in property whereas our semantics does not imply subset minimality of configurations as can be seen from Example 3. Semantics that are close to the disjunctive stable model semantics but allow non-minimal models have been proposed by Lifschitz and Woo [LW92] as well as Herre and Wagner [HW97]. However, these two approaches differ from ours as they use an exclusive reading of the disjunction. This means that, for example, a rule of the form $a \mid b \leftarrow$ leads to two models where a and b are true, respectively. On the other hand, in **CRL** an inclusive interpretation closer to classical disjunction is employed where a rule $a \mid b \leftarrow$ yields three configurations $\{a\}$, $\{b\}$, $\{a, b\}$.

Notice that our notion of configurations is more restricted than that of classical models. This means that configuration rules cannot be understood directly as a set of clauses with the classical models of the clauses as the configurations: the empty set is the unique valid configuration for $\{a \leftarrow b\}$ but seen as a clause it has a model where both a and b are true. In fact, our notion of a valid configuration of a set of rules coincides with the notion of a possible model introduced by Sakama and Inoue [SI94] for disjunctive programs. There is also an interesting correspondence to stable models of normal (non-disjunctive) programs. We present a mapping of **CRL** to normal logic programs where valid configurations for a set of rules correspond to stable models of the resulting logic program.

Given a set of rules R in **CRL** the corresponding normal logic program is constructed as follows. The requires-rules R_{r} are taken as such. The incompatibility-rules R_{i} are mapped to logic program rules with the same body but a head f and a new rule $f' \leftarrow \mathrm{not}(f'), f$ is included where f, f' are new atoms not appearing in R. For each choice-rule

$$a_1 \mid \cdots \mid a_l \leftarrow b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n)$$

in R_{c} a rule $f \leftarrow \hat{a_1}, \ldots, \hat{a_l}$ is included together with the rules, for all $i = 1, \ldots, l$,

$$\begin{aligned} a_i &\leftarrow \mathrm{not}(\hat{a_i}), b_1, \ldots, b_m, \mathrm{not}(c_1), \ldots, \mathrm{not}(c_n) \\ \hat{a_i} &\leftarrow \mathrm{not}(a_i) \end{aligned}$$

where $\hat{a_1}, \ldots, \hat{a_l}$ are new atoms. Now the stable models of the program provide the valid configurations for the rules. The close correspondence implies that an implementation of the stable model semantics can be used as an implementation of the configuration tasks. The correspondence between stable models and configurations reveals also a close relationship between the possible model semantics of disjunctive programs [SI94] and the stable model semantics of normal programs indicating, e.g., that this kind of a disjunctive semantics can be realized using an implementation of the stable model semantics for non-disjunctive programs.

5 Complexity Issues

In this section we consider the complexity of the key configuration tasks and show that from a knowledge representation point of view our configuration language is more easily maintainable than propositional logic. We study the following tasks. By **C-SAT** we denote the problem of deciding whether a configuration satisfies a set of rules, by **EXISTS** the problem of determining whether there is a valid configuration for a set of rules and by **QUERY** the problem of deciding whether there is a valid configuration C for a set of rules satisfying a set of requirements Q ($C \models Q$).

First, we observe that deciding **C-SAT** can be done in polynomial time. Second, we note that checking whether a set of atoms is a valid configuration for a set of rules can be achieved in polynomial time. This is because given a set of rules and a candidate configuration, the reduct can be computed in linear time and, similarly, the unique least model of a set of Horn clauses can be computed in linear time [DG84]. This also implies that the major computational tasks in configuration using our semantics are in **NP**.

For **EXISTS** and **QUERY** we consider subclasses of the configuration rule language **CRL**. For example, $\mathbf{CRL_r}$ is the subset where only requires-rules are allowed, $\mathbf{CRL_{rd}}$ permits additionally default negations and $\mathbf{CRL_{rci}}$ admits requires-rules, choice-rules and incompatibility-rules.

It turns out that we can add incompatibility-rules to requires-rules and still have polynomial algorithms for the configuration tasks. This is due to the fact that this subclass essentially corresponds to Horn clauses for which polynomial decision methods exist [DG84].

A language containing choice-rules, $\mathbf{CRL_{rc}}$, allows polynomial algorithms for **EXISTS**. This can be shown by transforming the disjunctions in the heads of rules to conjunctions. The result of this transformation is effectively a set of Horn clauses, whose minimal model can be shown to be a valid configuration of the original rule set. However, **QUERY** is **NP**-complete for $\mathbf{CRL_{rc}}$. This can be established using the following reduction from propositional satisfiability. Propositional clauses are mapped to configuration rules with negative literals forming the body of the rule and positive the head. For each clause without positive literals, the same head f is used where f is a new atom. As

Language	**C-SAT**	**EXISTS**	**QUERY**
$\mathbf{CRL_r}$	Poly	Poly	Poly
$\mathbf{CRL_{ri}}$	Poly	Poly	Poly
$\mathbf{CRL_{rc}}$	Poly	Poly	**NP**-compl.
$\mathbf{CRL_{rd}}$	Poly	**NP**-compl.	**NP**-compl.
$\mathbf{CRL_{rci}}$	Poly	**NP**-compl.	**NP**-compl.
$\mathbf{CRL_{rcid}}$	Poly	**NP**-compl.	**NP**-compl.

Table 1: Complexity results of configuration tasks

the query, the rule $f' \leftarrow f$ is employed where f' is yet another new atom. Now it can be shown that the set of clauses has a propositional model iff there is a valid configuration satisfying the query $f' \leftarrow f$ for the corresponding set of rules.

If incompatibility-rules are added to $\mathbf{CRL_{rc}}$, the decision problems **EXISTS** and **QUERY** become **NP**-complete. This is because propositional satisfiability can be reduced to finding a valid configuration by treating propositional clauses as configuration rules (negative literals form the body of the rule and positive the head). It can be shown that the set of clauses has a propositional model iff the corresponding set of rules has a valid configuration.

Adding default negations to $\mathbf{CRL_r}$ turns the decision problems **EXISTS** and **QUERY NP**-complete as we essentially then have the stable model semantics of normal logic programs. The discussion is summarized in Table 1. These complexity results can also be established from the complexity of the possible model semantics [SI94, EG95].

We end the section with an argument that from a knowledge representation point of view our rule-based language is more attractive than propositional logic in configuration applications. We aim to show that configuration knowledge expressed by the rule-based language cannot be captured in an easily maintainable and updatable way using propositional logic.

One possibility of representing valid configurations for a set of configuration rules R with propositional logic is to provide a set of propositional formulae T such that an R-valid configuration can be obtained as formulae derivable from T. However, there are difficulties because the set of derivable formulae is unique for each set of premises but a set of rules can have multiple valid configurations. This leads to the idea that configurations can be represented by models of a set of formulae [Kle96]. Interestingly, it turns out that this is not possible if we insist that the propositional representation is easily maintainable. One of the key properties of such a representation is its modularity, meaning that small local changes in the configuration model lead to small local changes in the corresponding set of formulae. For example, a simple update like adding a new fact into the configuration model should result in a local change in the propositional representation related to the new fact but not affecting the other parts of the representation of the configuration model. We can show under mild assumptions on the notion of modularity that no such representation exists.

We say that configurations are *modularly representable* by propositional models iff for every set of configuration rules R there is a set of propositional formulae $\mathrm{T}(R)$ such that for every set of requires-rules F with empty bodies (facts) there is a set of propositional formulae $\mathrm{T}'(F)$ such that C is a valid configuration of $R \cup F$ iff there is a model of $\mathrm{T}(R) \cup \mathrm{T}'(F)$ agreeing with C. A propositional model M is said to agree with a configuration C iff for each atom a in the configuration language, $a \in C$ iff a is true in M. Hence, the corresponding propositional language can contain additional atoms but for the atoms in the configuration rules, the models should correspond to the configurations. Our notion of modular representation means that configuration rules can be represented independent of the basic facts representing, e.g., requirements so that a change in the facts does not lead to a change in the propositional representation of the configuration rules.

Theorem 5.1 *Configurations cannot be modularly represented by propositional models.*

Proof. This result is due to the difficulties in capturing groundedness using propositional models and can be demonstrated with a very simple counter-example. Consider the set rules $R = \{c \leftarrow b\}$ and assume that its configurations can be modularly represented by propositional models. Hence, there is a set of formulae $\mathrm{T}(R)$ such that all atoms a in the configuration language are false in every model of $\mathrm{T}(R)$ as R has the empty set as its unique valid configuration. This implies that for all such atoms a, $\neg a$ is a logical consequence of $\mathrm{T}(R)$. Consider now a set of facts $F = \{b \leftarrow\}$. The configuration model $R \cup F$ has a unique valid configuration $\{b, c\}$ which contradicts the fact that in all models of $\mathrm{T}(R) \cup \mathrm{T}'(F)$, both b and c are false. Thus, valid configurations cannot be modularly represented by propositional models. □

6 Previous Work on Product Configuration

Several configurators and models of product configuration tasks are based on constraint satisfaction problem

(CSP) solving. A CSP consists of a set of *variables*, a set of possible values for each variable, called the *domain* of the variable, and a set of *constraints*. A constraint specifies the allowed combinations of values for the variables that the constraint refers to. This is done by specifying the allowed subset of the cartesian product of the domains of the variables. A *solution* to a CSP is an *assignment* of values to all variables such that the constraints are satisfied.

A CSP can be represented in **CRL** by mapping it to a set of rules in **CRL** as follows: introduce for each pair $\langle variable, possible value of the variable \rangle$ a distinct atom. Each variable and its domain is represented as an exclusive choice whose head consists of all the possible pairs for that variable and whose body is empty. The constraints can be represented using incompatibility rules to prevent the non-allowed combinations of values, provided that the domains are finite. It is easy to see that each valid configuration provides a solution to the CSP and vice versa.

However, not all constructs of **CRL** have a convenient CSP-representation. Consider an inclusive choice rule without a body, which basically represents choosing a subset of a set of atoms. A straightforward CSP-representation for this would use one distinct variable for each alternative atom, each with domain $\{true, false\}$, and a constraint that not all the variables have the value $false$. Representing constraints on this type of formulation may be awkward as some variables represent only parts of one inclusive choice in the original problem while other variables could be used to encode entire (exclusive) choices. This complicates the maintenance of knowledge as the connection between the choices and the resulting variables, domains, values and constraints is not self-evident.

Similarly to the propositional logic, the CSP formalism cannot modularly capture groundedness of a configuration. This can be shown in the same way as in Theorem 5.1. We use the following notion of modularity. Configurations are defined to be modularly representable by solutions to a CSP iff for every set of rules R there is a CSP representation $\mathrm{T}(R)$ such that for every set of requires-rules F with empty bodies (facts) there is a *simple update* $\mathrm{U}(\mathrm{T}(R), F)$ to the CSP representation such that C is a valid configuration for $R \cup F$ iff there is a solution of $\mathrm{U}(\mathrm{T}(R), F)$ agreeing with C. A simple update is defined as one of the following updates: either add or remove a set of allowed value tuples, either add or remove a set of constraints and either add or remove a set of variables. Note that simple updates also allow either adding or removing sets of values from the domains if these are interpreted as adding or removing constraints. A solution S to a CSP is said to agree with a configuration C iff (i) there is a distinct variable with the domain $\{true, false\}$ in S for each propositional atom occurring in R and (ii) for each atom a in the configuration language, $a \in C$ iff the variable representing a has the value *true* in S. Note that (i) is merely required for convenience of interpreting the solution and is not a real restriction, since the representation may additionally contain arbitrary variables, values and constraints.

This notion of modular representation means that configuration rules should be represented as a CSP independent of the basic facts so that a change in the facts does not lead to a change involving both additions and removals of tuples, constraints, variables or values. We note that a rule set can modularly represent the solutions of a CSP. This can be easily seen from the translation from CSP to **CRL** given above.

Theorem 6.1 *Configurations cannot be modularly represented by solutions to CSP.*

Proof. Consider the set rules $R = \{c \leftarrow b\}$ and assume that its configurations can be modularly represented by solutions to a CSP. Hence, there is a CSP $\mathrm{T}(R)$ such that in all the solutions of $\mathrm{T}(R)$ the variables representing atoms b and c in the configuration language have the value $false$ as R has the empty set as its unique valid configuration. Consider now a set of facts $F = \{b \leftarrow\}$. The configuration model $R \cup F$ has a unique valid configuration $\{b, c\}$. This means that $\mathrm{U}(\mathrm{T}(R), F)$ must not have a solution in which variables encoding b and c have the value $false$. In addition, $\mathrm{U}(\mathrm{T}(R), F)$ must have at least one solution in which the atoms encoding b and c have the value *true*. It can be shown that none of the simple updates can both add solutions and remove them, which contradicts the assumption. □

The dynamic CSP formalism (DCSP) [MF90] was defined to provide a meta-level representation and reasoning mechanism for expressing conditions under which a variable needs to have a value, i.e. conditions on *activity* of variables. This is achieved by adding *activity constraints* to a regular CSP. In contrast to DCSP, we provide a uniform way to represent activity and other constraints and to reason about them. The DCSP approach does not include activity constraints that have a disjunction in the head or negated conditions in the body, which is possible in **CRL**.

As shown above, **CRL** allows representing modularly and compactly some forms of typical configuration knowledge. This is not the case for the other discussed formalisms. Therefore, we consider **CRL** a potentially more adequate configuration knowledge representation formalism from the knowledge management point of view (with the caveat that some constructs now treated as short-hand notation need to be added to the basic formalism).

The generative CSP (GCSP) [HS93] approach introduces first-order constraints on activities of variables, on variable values and on resources. Constraints using arithmetic are also included. *Resources* are aggregate functions on intensionally defined sets of variables. They may restrict the set of variables active in a solution or generate new variables into a solution, thus providing a justification for the variables. In addition, a restricted form of DCSP activity constraints is used to provide justifications for activity of variables. As discussed above, **CRL** allows more expressive activity constraints and a uniform representation of activity and other constraints. However, **CRL** clearly does not allow first-order rules, arithmetic or resource functions.

As a final remark, our approach fits broadly within the framework of constructive problem solving (CPS) [Kle96]. In that approach, the configurations are characterized as (possibly partial) Herbrand models of a set of sentences in an appropriate logic language. The CPS approach does not require that a configuration is grounded. The need for meta-level minimality criterion is mentioned, though.

7 Conclusions and Future Work

We have presented a formal model of the interactions of knowledge elements in the configuration domain. In our formalization all the elements of a valid configuration must be grounded in the rules of the configuration model. This property is an important characteristic whose formal definition has been largely neglected.

The formal model is closely related to well-known nonmonotonic formalisms such as stable model semantics [GL88] and possible model semantics [SI94]. The semantics is defined using a simple transformation operator allowing a straightforward fix-point characterization. We show that valid configurations cannot be modularly represented as models of propositional logic or as solutions to a constraint satisfaction problem.

The basic problems of the validity of a configuration with respect to a configuration model and whether a configuration satisfies a set of customer requirements are polynomially decidable. This implies that other relevant decision problems are in **NP**. These results are due to the groundedness of a configuration not being captured by a minimality criterion on the configurations but in terms of derivability from a set of Horn clauses. The tractability of the basic decision problems is important for practical configuration problems.

There are indications that the proposed formal model provides a basis for solving practically relevant product configuration problems. As shown above, implementations of stable model semantics can be used for the decision problems of our model. There are efficient implementations of stable models semantics capable of handling tens of thousands of ground rules, such as Smodels [NS96]. Compiling a practically relevant configuration model from a high level representation into ground rules would seem to generate rule sets of approximately that size. Furthermore, it may be possible to develop a more efficient algorithm that avoids the overhead incurred by the additional atoms and the loss of information on the structure of the rules resulting from the translation. Devising such an algorithm is an interesting subject of further work. A practically important task would be to identify additional syntactically restricted but still useful subsets of the language that would allow more efficient computation than the general cases discussed here.

It should be noted that the model does not adequately cover all the aspects of product configuration. Further work should include generalizing the rules to the first-order case and adding arithmetic operators to the language. These extensions are needed to conveniently represent resource constraints, attributes, structure and connections of components. Another important extension would be to define the notion of an optimal configuration (such as subset minimal, cardinality minimal or resource minimal configuration) and to analyze the complexity of optimality-related decision problems. Interactive product configuration where user makes hard decisions and computer only tractable ones would be facilitated by defining a polynomially computable approximation semantics whose relation to the semantics presented here would be similar to that of the well-founded semantics [VGRS91] to the stable model semantics.

Acknowledgements

The work of the first author has been supported by the Helsinki Graduate School in Computer Science and Engineering (HeCSE) and the Technology Development Centre Finland. The work of the second author has been supported by the Academy of Finland.

References

[BD95] S. Brass and J. Dix. Disjunctive semantics based on partial and bottom-up evaluation. In *Proceedings of the 12th International Conference on Logic Programming*, pages 199–213, Tokyo, June 1995. The MIT Press.

[CGS+89] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—an approach to domain-independent construction. In *The Second International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE-89*, volume 2, pages 866–874. ACM, 1989.

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[EG95] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15:289–323, 1995.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[HJ91] M. Heinrich and E.W. Jungst. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings of the seventh IEEE conference on Artificial Intelligence Applications*, pages 257–264, 1991.

[HS93] Alois Haselböck and Markus Stumptner. An integrated approach for modelling complex configuration domains. In *Proceedings of the 13th International Conference on Expert Systems, AI, and Natural Language*, Avignon, May 1993.

[HW97] Heinrich Herre and Gerd Wagner. Stable models are generated by a stable chain. *Journal of Logic Programming*, 30(2):165–177, February 1997.

[Kle96] R. Klein. A logic-based description of configuration: the constructive problem solving approach. In B. Faltings and E. Freuder, editors, *Configuration—Papers from the 1996 AAAI Fall Symposium. Technical Report FS-96-03*, pages 111–118. AAAI Press, 1996.

[LW92] Vladimir. Lifschitz and Thomas Y.C. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 603–614, Cambridge, MA, USA, October 1992. Morgan Kaufmann Publishers.

[McD82] J. McDermott. R1: a rule-based configurer of computer systems. *Artificial Intelligence*, 19(1):39–88, 1982.

[MF89] Sanjay Mittal and Felix Frayman. Towards a generic model of configuration tasks. In *Proceedings of IJCAI 89*, pages 1395–1401. AAAI, 1989.

[MF90] Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pages 25–32. AAAI, MIT Press, 1990.

[NS96] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[Prz95] T.C. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence, Special Issue on Disjunctive Programs*, 14:323–357, 1995.

[SI94] C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13:145–172, 1994.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.

[TSMS96] Juha Tiihonen, Timo Soininen, Tomi Männistö, and Reijo Sulonen. State-of-the-practice in product configuration—a survey of 10 cases in the Finnish industry. In T. Tomiyama, M. Mäntylä, and S. Finger, editors, *Knowledge Intensive CAD*, volume 1, pages 95–114. Chapman & Hall, 1996.

[VGRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.