Towards an Implementation of a Multi-Context System Framework

Tarek R. Besold and Stefan Mandl

University of Erlangen-Nuremberg,
Chair of Computer Science 8: Artificial Intelligence,
D-91058 Erlangen, Germany
sntabeso@i8.informatik.uni-erlangen.de
stefan.mandl@informatik.uni-erlangen.de

Abstract. This system description provides (after a limited formal introduction to the topic) an overview of our proof-of-concept multi-context system framework implementation, lining out the main functionalities and working principles. Moreover, two basic techniques for reducing computational complexity when searching for equilibria of a multi-context system are proposed.

1 Introduction

One of the basic problems in knowledge representation and knowledge engineering is the impossibility of writing globally true statements about realistic problem domains. Multi-context systems (MCS) are a formalization of simultaneous reasoning in multiple contexts. Different contexts are inter-linked by bridge rules which allow for a partial mapping between formulas/concepts/information in different contexts. Recently there have been a number of investigations of MCS reasoning (for instance, see [1] or [2]), with [3] and [4] being two of the latest contributions. In the former one, the authors describe reasoning in multiple contexts that may use different logics locally. In the latter one, the concept of MCS reasoning is extended to also integrate sub-symbolic contexts, and an algorithm for finding the equilibria of an MCS is presented.

In this paper, we want to describe how we designed and implemented a proof-of-concept MCS framework software, based on results from [3] and [4].

2 (Very) Crisp Introduction to Multi-Context Systems

As an introduction to the topic, we restate the key definitions given in [3]: First, the concept of *logic*[1] is defined in terms of input-output conditions.

[1] As in the following, no information containing satisfiability or inference rules within the corresponding logics will be given, also the denomination "pre-logic" would be justifiable. For the sake of consistency we will keep the original naming calling it a "logic".

Definition 1. *A logic* $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$ *is composed of the following components:*

1. $\mathbf{KB}_L$ *is the set of well-formed knowledge bases of* L*. We assume each element of* $\mathbf{KB}_L$ *is a set.*
2. $\mathbf{BS}_L$ *is the set of possible belief sets,*
3. $\mathbf{ACC}_L : \mathbf{KB}_L \mapsto 2^{\mathbf{BS}_L}$ *is a function describing the "semantics" of the logic by assigning to each element of* $\mathbf{KB}_L$ *a set of acceptable sets of beliefs.*

Given several logics, bridge rules are used to translate between the logics.

Definition 2. *Let* $L = \{L_1, \ldots, L_n\}$ *be a set of logics. An* L_k *-bridge rule over* $L, 1 \leq k \leq n$*, containing* m *conditions, is of the form*

$$s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \mathbf{not}(r_{j+1} : p_{j+1}), \ldots, \mathbf{not}(r_m : p_m) \tag{1}$$

where $j \leq m$*,* $1 \leq r_k \leq n$*,* p_k *is an element of some belief set of* L_{r_k} *and* s *is a syntactically valid element of a knowledge base from* $\mathbf{KB}_k$*,*[2] *and for each* $kb \in \mathbf{KB}_k : kb \cup \{s\} \in \mathbf{KB}_k$*.*

A configuration of logics and bridge rules comprises a multi-context system.

Definition 3. *A multi-context system* $M = (C_1, \ldots, C_n)$ *consists of a collection of contexts* $C_i = (L_i, kb_i, br_i)$*, where* $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ *is a logic,* kb_i *a knowledge base (an element of* $\mathbf{KB}_i$ *), and* br_i *is a set of* L_i*-bridge rules over* $\{L_1, \ldots, L_n\}$*.*

A belief state is the combination of the belief sets of all contexts of the MCS.

Definition 4. *Let* $M = (C_1, \ldots, C_n)$ *be a MCS. A belief state is a sequence* $S = (S_1, \ldots, S_n)$ *such that each* S_i *is an element of* $\mathbf{BS}_i$*.*

Now we can clarify the applicability of a bridge rule in the context of a belief state: We say a bridge rule r of form (1) is applicable in a belief state $S = (S_1, \ldots, S_n)$ iff for $1 \leq i \leq j : p_i \in Th(S_{r_i})$ and for $j + 1 \leq k \leq m : p_k \notin Th(S_{r_k})$.[3]

A belief state is an equilibrium if the consequences of all applicable bridge rules are given or derivable, hence each context has an acceptable belief set given the belief sets of the other contexts.

Definition 5. *A belief state* $S = (S_1, \ldots, S_n)$ *of* M *is an equilibrium iff, for* $1 \leq i \leq n$*, the following condition holds:*

$$S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) | r \in br_i \text{ applicable in } S\}).$$[4]

[2] In contrast to [3] where a similar constraint concerning the nature of s is imposed only implicitly.

[3] $Th(\cdot)$ indicates the deductive closure, i. e. $Th(\Theta) = \{\varphi \in \mathbb{L} \mid \Theta \models \varphi\}$, where Θ is a set of formulae, φ is a formula, and $\mathbb{L}$ is a logic.

[4] Please note that, if r is a bridge rule of the form $a \leftarrow \ldots$, then $head(r) = a$

Equilibria are a fundamental concept of multi-context systems as they make the introduction of a notion of semantics for MCS possible, and under certain circumstances exhibit properties analogue to those of fixpoints as means of semantics: In [3], a restricted class of MCS is introduced, for which the authors present a reduction similar to the Gelfond/Lifschitz reduction for logic programs. For this class of "reducible MCS", the notion of "grounded equilibria" is given, following ideas and terminology from logic programming. Brewka and Eiter finally define a well-founded semantics for reducible MCS by constructing an operator involving the concepts of reducibility and grounded equilibria, and using a fixpoint argument in the style of the Knaster-Tarski theorem.

For the implementation of the proof-of-concept MCS framework, we used a generalized account of the just outlined theory, which also allows for the incorporation of sub-symbolic contexts of reasoning. We will now briefly restate the relevant concepts (introduced in [4], where also a more thorough discussion of the now following account may be found), which may be understood as direct generalizations of the above stated ideas.

Definition 6. *A* reasoner *is a 5-tuple* $R = (\mathbf{Inp}_R, \mathbf{Res}_R, \mathbf{ACC}_R, \mathbf{Cond}_R, \mathbf{Upd}_R)$ *where*

1. $\mathbf{Inp}_R$ *is the set of possible* inputs *to the reasoner.*
2. $\mathbf{Res}_R$ *is the set of possible* results *of the reasoner*
3. $\mathbf{ACC}_R : \mathbf{Inp}_R \mapsto 2^{\mathbf{Res}_R}$ *defines the actual reasoning, assigning each input a set of results in a decidable manner.*
4. $\mathbf{Cond}_R$ *is a set of decidable conditions on inputs and results:*

$$cond_R : \mathbf{Inp}_R \times \mathbf{Res}_R \mapsto \{0, 1\}.$$

5. $\mathbf{Upd}_R$ *is a set of update functions for inputs:*

$$upd_R : \mathbf{Inp}_R \mapsto \mathbf{Inp}_R\,.$$

Please note that in contrast to Definition 1 we do not make any assumption about the form of the input to the reasoner.

The following definitions adapt the basic concepts of multi-context reasoning given in Definitions 2 to 5 for the use with reasoners as of Definition 6.

Definition 7. *Let* $R = \{R_1, \ldots, R_n\}$ *be a set of reasoners. An* R_k *-bridge rule over* $R, 1 \leq k \leq n$*, containing* m *conditions, is of the form*

$$\begin{aligned} u \leftarrow\ & (r_1 : c_1), \ldots, (r_j : c_j), \\ & \mathbf{not}(r_{j+1} : c_{j+1}), \ldots, \mathbf{not}(r_m : c_m) \end{aligned} \tag{2}$$

where $j \leq m$*,* $1 \leq r_k \leq n$ *and* c_k *is a condition of inputs and results of some* R_{r_k} *and* u *is an element of the* $\mathbf{Upd}_k$*.*

Analogously to Definition 3 we now define a generalized MCS.

Definition 8. *A generalized multi-context system* $M = (C_1, \ldots, C_n)$ *consists of a collection of contexts* $C_i = (R_i, inp_i, br_i)$*, where* $R_i = (\mathbf{Inp}_i, \mathbf{Res}_i, \mathbf{ACC}_i, \mathbf{Cond}_i, \mathbf{Upd}_i)$ *is a reasoner,* inp_i *an input (an element of* $\mathbf{Inp}_i$ *), and* br_i *is a set of* R_i*-bridge rules over* $\{R_1, \ldots, R_n\}$ *as of equation (2).*

Belief states also have to be adapted.

Definition 9. *Let* $M = (C_1, \ldots, C_n)$ *be a generalized MCS. A generalized belief state is a sequence* $S = (S_1, \ldots, S_n)$ *such that each* S_i *is of the form* (inp_i, res_i) *with* $inp_i \in \mathbf{Inp}_i$ *and* $res_i \in \mathbf{Res}_i$*.*

We say a bridge rule r of form (2) is applicable in a generalized belief state $S = (S_1, \ldots, S_n)$ iff for $1 \leq i \leq j : c_i(inp_i, res_i) = 1$ in S_i and for $j + 1 \leq k \leq m : c_k(inp_k, res_k) = 0$ in S_k. Now we prepare for the concept of equilibrium in the generalized setting.

Definition 10. *The set of (context local) update functions with respect to a corresponding element* S_i *of a belief state* S *is given by*

$$\mathbf{US}_i(MCS, S) = \{head(r) | r \in br_i \text{ applicable in } S\},$$

where br_i *denotes the set of bridge rules of* S_i*'s corresponding context* C_i*.*

In general, a set of update functions may yield different results when the functions are applied multiple times or in different orders. We do not allow such sets of update functions.

Definition 11. *An applicable set of update functions* $\mathbf{US}_i(MCS, S)$ *is stationary for an input* inp_i *iff the following two conditions hold:*

- $\forall u \in \mathbf{US}_i(MCS, S) : u(inp_i) = u^m(inp_i)$ *for* $m \geq 1$ *(idempotency)*
- $\forall u, u' \in \mathbf{US}_i(MCS, S) : u(u'(inp_i)) = u'(u(inp_i))$ *(commutativity)*

Definition 12. *The update of a belief state element* S_i *of a belief state* S*, with respect to a set of update functions* $\mathbf{US}_i$ *with* k *elements, is given by*

$$u_1(u_2(\ldots u_k(inp_i)\ldots))$$

if $\mathbf{US}_i$ *is stationary for* inp_i*, and undefined otherwise.*

Please note that stationarity is only required for the set of update functions that is actually applied to belief state elements at a time. We now can give the definition of the generalized concept of equilibrium.

Definition 13. *A generalized belief state* $S = ((inp_1, res_1), \ldots, (inp_n, res_n))$ *of* M *is an equilibrium iff, for* $1 \leq i \leq n$*, the following condition holds:*

$$update(inp_i, \mathbf{US}_i) = inp_i \text{ and } res_i \in ACC(inp_i)$$

where $update(inp_i, \mathbf{US}_i)$ *denotes the update of* S_i *with respect to* US_i*, which in turn has to be stationary for the corresponding* inp_i*.*

The remainder of this section describes a procedure to compute all equilibria of a finite MCS, based on complete enumeration, which served as basis for our proof-of-concept implementation.

Definition 14. *An MCS $M = (C_1, \ldots, C_n)$ is said to be finite, iff for $1 \leq i \leq n$, following condition holds: $|ACC(inp_i)| < \infty$ and $|br_i| < \infty$.*

For the implementation, we consider finite MCS only.

Definition 15. *Let* **Br** *be a set of n bridge rules of an MCS. A bridge rule model is an assignment* $\mathbf{Br} \mapsto \{0,1\}^n$ *that represents for each bridge rule in* **Br** *whether it is active or not.*

Proposition 1. *For each equilibrium there is exactly one bridge rule model.*

For a given bridge rule model and an MCS we first apply all the bridge rules activated in the bridge rule model yielding $inp'_1 ... inp'_n$. Then we compute the set of results for each context i given inp'_i by applying $ACC(inp'_i)$, yielding a set of results res_i^j for each i, being of finite cardinality as MCS was said to be finite. Thus, testing whether (inp_i, res_i^j) is an equilibrium for all j, we obtain the set of equilibria for the given bridge rule model. Iterating the procedure over the (finite) set of all bridge rule models and joining the resulting sets of equilibria finally yields the set of all equilibria.

Definition 16. *Given an MCS with a (global) set of bridge rules $br = \bigcup_i br_i$. A set of bridge rules $br_j \subseteq br$ is called update-monotonic iff for all belief states S, S' the following condition holds:*

$$S' = update(MCS, S) \Rightarrow VC(MCS, S) \subseteq VC(MCS, S')$$

where

$$VC(MCS, S) = \bigcup_i \{cond_i \in R_i | cond_i(inp_i, res_i) = 1\}$$

and $update(MCS, S)$ is the (global) update over all $S_i \in S$.

As bridge rules in the update-monotonic subset of bridge rules of the MCS are guaranteed to remain active after any update, the update-monotonic bridge rules that are initially active in the MCS when searching for equilibria have to be active in any equilibrium. Hence, when iterating over all bridge rule models, only those bridge rule models that comply with the initially active update-monotonic bridge rules have to be considered.

Proposition 2. *For finite MCS, the above sketched algorithm (for a pseudo code representation see Algorithm 1) is both complete and correct.*

3 Refinements of the Basic Algorithm

In the following, we present two refinements – mainly used for reducing computational complexity – we added to the MCS algorithm from [4].

Input: $MCS = (C_1, \ldots, C_n)$ (a finite multi-context system)
Output: answer
answer $\leftarrow$ {}
$br \leftarrow \bigcup_{1 \leq i \leq n} br_i$
$ums \leftarrow$ UpdateMonotonicBridgeRules(MCS,br)
$cand \leftarrow \{brm \in 2^{br} | ums \subseteq brm\}$
for $brm \in cand$ **do**
  $S \leftarrow ((inp_1, res_1), \ldots, (inp_n, res_n))$
  for $b \in brm$ **do**
    $S \leftarrow$Apply$(head(b), S)$
  for $1 \leq i \leq n$ **do**
    $S[i] \leftarrow$(inp$(S[i])$, $\mathbf{ACC}_i$(inp$(S[i])$)
  for $1 \leq i_1 \leq |$res$(S_1)|$, $\ldots$, $1 \leq i_n \leq |$res$(S_n)|$ **do**
    $S^* \leftarrow$ ((inp(S_1),res$(S_1)[i_1]$),$\ldots$, (inp(S_1),res$(S_1)[i_n]$))
    if Equilibrium(S^*) **then**
      answer$\leftarrow$ answer$\cup\{S^*\}$

Algorithm 1: Algorithm for computing all equilibria of an MCS

3.1 The Notion of Conflicting Bridge Rules

The first refinement comprises a method for pruning out some bridge rule models (BRMs) already before they are falsely considered as possible equilibria.

We exploit the (within every context locally valid) demand for consistency in order to reduce the equilibria search space via "conflicting bridge rules":

Definition 17. *Two bridge rules $br_i, br_j \in br$ (where br is a set of bridge rules of an MCS) are called i-conflicting if applying br_i would directly inhibit the application of br_j due to a condition $c_{jk} \in body(br_j)$ which contains (where applicable) "$\neg head(br_i)$" or "not $head(br_i)$" (or any other statement contradicting $head(br_i)$).*

This allows for an extension of the algorithm: For a given set of bridge rules br, we can initially compute, for every $br_i \in br$, the corresponding set of i-conflicting bridge rules in br. As every active bridge rule $br_i \in br$ may directly inactivate all i-conflicting bridge rules, we may further reduce the number of BRMs tested on being an equilibrium by setting all i-conflicting bridge rules to 0, when setting br_i to 1.

The additional effort spent on initially computing the conflicting bridge rules within the set br in most cases pays off, as in total $\sum_{m=1}^{C} 2^{n-(1+m)}$ (n the total number of bridge rules of the MCS, C the total number of i-conflicting bridge rules for all $br_i \in br$) BRMs are pruned out. Hence, at least $(\sum_{m=1}^{C} 2^{n-(1+m)}) \cdot N$ (N the number of contexts of the MCS) reasoner calls less have to be performed. An upper boundary for the extra costs of computing the conflicting bridge rules within br is given by $n \cdot l$ (n the total number of bridge rules of the MCS, l the

maximum of the numbers of conditions in the body of a bridge rule from br) comparisons between bridge rule heads and bridge rule conditions from bridge rules within the set br.[5]

3.2 Constraints on Bridge Rules

Moreover, the wish to reduce some complexity by making knowledge of the combinatorical structure of bridge rules (that is already implicitly contained in the MCS) explicit may arise. In order to do so, constraints on bridge rules or groups of bridge rules may be used, resulting in a formalism to impose constraints on the BRMs to be considered.

The constraint formalism shall offer possibilities to group bridge rules into classes and impose cardinality restrictions on these classes up to uniqueness of a rule as a representant of a certain rule class. It may be used to model interconnectedness and hierarchy between the classes of bridge rules by establishing a concept of subclasses and superclasses, as well as introducing complementary classes. Moreover, features have been added to provide some kind of rudimentary inference mechanism amongst bridge rules/bridge rule classes.[6]

1. $X.maxCard(numb)$: $class_1.maxCard(a)$, with $0 \leq a \leq max$, where max is the total number of bridge rules in the model, expresses that only BRMs in which at most a bridge rules from the class $class_1$ are active shall be considered.
2. $X.minCard(numb)$: Analogously.
3. $X.unique$: $class_1.unique$ expresses that only BRMs in which exactly one bridge rule from the class $class_1$ is active shall be considered.
4. $X.bundled$: $class_1.bundled$ expresses that only BRMs in which all bridge rules from the class $class_1$ are active, or BRMs in which all bridge rules from the class $class_1$ are inactivated, shall be considered.
5. $X.superClass(Y)$: $class_1.superClass(class_2)$ expresses that $class_1$ has $class_2$ as a superclass, i.e. $class_1 \subset class_2$.
6. $X.subClass(Y)$: Analogously.
7. $X.complementary(Y)$: $class_1.complementary(class_2)$ expresses that only BRMs in which bridge rules from $class_1$ or bridge rules from $class_2$, but not bridge rules from both classes at a time, are active shall be considered.
8. $X.oneOfImplicatesExactlyOne(Y)$: $class_1.oneOfImplicatesExactlyOne(class_2)$ expresses that only BRMs in which, whenever a bridge rule of $class_1$ is active, exactly one bridge rule out of $class_2$ is active, shall be considered.
9. $X.oneOfImplicatesSome(Y)$: Analogously.
10. $X.oneOfImplicatesAll(Y)$: Analogously.
11. $X.entireClassImplicatesExactlyOne(Y)$: $class_1.entireClassImplicatesOne(class_2)$ expresses that only BRMs in which, whenever the bridge rules of $class_1$ are all active, exactly one bridge rule out of $class_2$ is active, shall be considered.

[5] As in general the comparisons between heads and conditions can be performed by magnitudes more time-efficient as a reasoner call (e.g. in many logical contexts the comparison may be performed by a simple syntax based matching between heads and conditions), the overall computational efficiency of the MCS will be improved.

[6] In the following using the implemented constraint language, where X and Y are variables which can be replaced by concrete classes, and $numb$ is a numeric integer variable.

12. $X.entireClassImplicatesSome(Y)$: Analogously.
13. $X.entireClassImplicatesAll(Y)$: Analogously.

The individual bridge rules are explicitly distributed over the different classes by labeling them with the class name, e. g. directly in the bridge rule base (by this establishing some kind of $instanceOf$ relation between the individual bridge rules and the classes they belong to: the class being the concept, the bridge rule the instance). Every unlabeled bridge rule is implicitly added to a generic class of bridge rules ($class_{generic}$) which is the superclass of all other classes. On $class_{generic}$, no constraints in form of properties as listed above may be imposed.

4 A Proof-of-concept MCS Framework Implementation

The proof-of-concept implementation is based on the principles of the aforementioned algorithm for finding the equilibria of an MCS presented in [4]. After some steps of reading the MCS specification, getting input and creating the local representations of the knowledge bases and the sets of bridge rules of the MCS, we generate all possible BRMs. Then for each BRM we perform the updates it indicates and check whether the result is an equilibrium or not. When doing so we also perform a test if the conditions of all applied bridge rules are really fulfilled, and the bridge rules have been applied justifiedly. According to the result of this tests, we add the BRM to the list of equilibria BRMs, or we directly proceed with the next bridge rule model.

Inter alia for the implementation of the constraint mechanism for bridge rules and bridge rule classes, we make use of the lparse/smodels combination (see [5] and [6]) as implementation of the stable model semantics for logic programmes. Both are at some points called as external components.

4.1 Infrastructure & Ex Ante Setup

As programming language for the first implementation[7] of the MCS framework we used Scala, a general purpose programming language.

In our MCS framework, an MCS is unequivocally defined by an MCS configuration file. In the file all contexts are represented. Having read the configuration file and built an internal representation of each context, the MCS framework sets up a list of initial knowledge or input bases, as well as a list of bridge rule bases, each associated with a context.

Moreover, if indicated in the invocation, the bridge rule constraint configuration file is processed. This file contains information concerning the grouping of bridge rules into different classes, the properties of individual bridge rule classes and the relations between different classes (see Section 3.2). For doing so, the different bridge rule classes explicitely have to be declared as such in the file, afterwards their properties and the relations between the classes may be named.

[7] In the meantime, a second implementation using Clojure has been built.

If this functionality shall be used, in the bridge rule bases every bridge rule individually has to be labelled with the name of the class it belongs to. Unlabelled bridge rules by default will be treated as belonging to $class_{generic}$.

After performing these steps, the BRMs which afterwards will be tested for representing an equilibrium are generated. This may be done by creating a bitstring representing a BRM for each possible combination of activation/inactivation of bridge rules. Thereby, in total 2^n (n the total number of bridge rules in the MCS) bitstrings are generated.

Due to performance considerations, we implemented another mechanism, exploiting – whenever indicated and possible – constraints on bridge rules and conflicting bridge rules. In this scenario, we are using the lparse/smodels answer set solver to generate the BRM bitstrings. This offers the possibility to directly include the information given in the constraints on the bridge rules into the model generation: smodels is used for generating a complete list of models of bridge rule combinations complying with the constraints stated via the constraints on bridge rules formalism, which then are transformed into BRM bitstrings. By this, no a priori contradicting bridge rule combinations are considered possible BRMs. At least for logical contexts we may also use the notion of conflicting bridge rules, further sparsening the field of BRMs. The bridge rule head bases and the bridge rule body bases are scanned for conflicting structures, adding information of any conflict found to the input for the answer set solver.

To start the equilibria mechanism, the MCS's knowledge bases, the bridge rule bases and the BRMs are handed over to the "$findEquilibria$" subroutine.

"$findEquilibria$" returns pairs of equilibria and corresponding BRMs. The output may be written out directly, or postprocessing steps may be performed.

4.2 The Equilibria Mechanism: "*findEquilibria*"

Given the MCS's knowledge bases, the bridge rules, a list of BRMs and the contexts "$findEquilibria$" lists the equilibria amongst the BRMs and the corresponding belief states of the respective contexts.

Given a BRM, the heads and bodies of the corresponding bridge rules, the contexts and the knowledge bases, "$findEquilibria$" performs all the updates which are indicated by the heads of bridge rules set to 1 in the given BRM. This is done by a subroutine called "$createReasonerInput''$. Then it invokes the corresponding reasoners (indicated by the information stored in the contexts), thereby performing a global reasoner call over all the (possibly updated) knowledge bases. If the reasoner calls yield consistent results for all knowledge bases, for every inactivated bridge rule "$findEquilibria$" has to test whether the bridge rule is correctly set to 0 in the BRM. To do so, it has to find in every inactivated bridge rule's body at least one condition that is not fulfilled. The same has to be done for all the bridge rules set to 1: All the conditions in every bridge rule body have to be fulfilled. If this tests may successfully be completed, the BRM represents an equilibrium for the given MCS, and the BRM, as well as the the belief state are linked and jointly added to the list of equilibria. Then, "$findEquilibria$" proceeds with the next BRM (if there is any).

As long as we are dealing with logical contexts for which we may use the lparse/smodels answer set solver as a reasoner, we may make the test for founded activation of the 1-bridge rules superfluous, by modifying the "*createReasoner-Input*" mechanism: For every knowledge base element a, demanded in a condition in the body of an activated bridge rule, we integrate a constraint ":- *not* a" into the answer set solver input (which in its result inhibits the generation of a model in that a is not present), and for every knowledge base element b, listed in an inhibitive condition, we add ":- b" (preventing the occurrence of models in which b is present). Thus, we assure that if a model is returned, for every 1-bridge rule the conjunction of the conditions in its body holds.

For the inactivated bridge rules this technique may not be applied: For a bridge rule set to 0 it is sufficient that only one of its conditions fails. Thus, doing as we did before with the 1-bridge rules would be by far too restrictive.

But nevertheless, as long as we are dealing with contexts allowing for smodels as a reasoner, and yielding unique reasoning results (i.e. only one model is returned when smodels is called as a reasoner), we may exploit some functionality of smodels when testing for founded inactivation of a 0-bridge rule: The following test will be based on comparing the total number of bridge rules in the MCS with the number of correctly activated/inactivated bridge rules. If both values are equal, the BRM represents an equilibrium of the MCS.

Given the valid results of the reasoner calls, we cycle through the representation of the contexts in the BRM. If a context's BRM part doesn't contain a 0, we may directly add the number of bridge rules of this context to the number of foundedly activated/inactivated bridge rules (founded activation has already been assured by the use of the smodels-optimization when creating the reasoner input for the context) and proceed with the next context's representation. Otherwise, we take the context's model, which was previously returned as result by the smodels call. For every inactivated bridge rule, we cycle through its body, and add every condition to a list corresponding to the conditions target context (e.g. for "$2 :$ *not* q" "*not* q" would be added to C_2's list). Having completely traversed the bridge rule's body, we add to every context's reasoning result a constraint, containing the conjunction of the elements of the context's list. E.g. for the list "{p, *not* q, r, s}" the constraint ": $-$ p, *not* q, r, s." would be added to the model previously returned by the smodels call. Having done so for all the contexts of the MCS, we perform another global reasoner call, invoking smodels for every single context, handing over the modified reasoning results of the earlier calls. If there is at least one context – with modified reasoning result as input – for which the second reasoner call returns a valid model (i.e. the conjunction constraint is not applied), we may augment the number of foundedly activated/inactivated bridge rules by one and proceed with the next bridge rule. If the models of all contexts with modified input are invalid, the 0-bridge rule has been inactivated unfoundedly, and the BRM does not represent an equilibrium.

Unfortunately, we have to perform this process for every single bridge rule, and cannot include all constraints from bridge rules into one global reasoner

call, as bridge rules which "overlap" in their conditions (e.g. one containing "$2 :\ not\ s$", the other containing "$2 :\ s$') may otherwise cause wrong results.

4.3 Identifying Self-sustaining Equilibria

For MCS consisting only of logical contexts using smodels as a reasoner, we added "$findSstEquilibria''$, used to list the self-sustaining[8] (s-st.) equilibria.

The "$findSstEquilibria$" function is given a BRM, the knowledge bases, the bridge rules and the contexts. The "$createReasonerInput$" subroutine is called with empty knowledge bases (again performing the smodels-optimization already mentioned), and the result is handed over to the contexts' reasoners. If a reasoner returns an invalid (e.g. inconsistent) reasoning result, the BRM does not represent an s-st. equilibrium of the given MCS.

In order to eliminate all equilibria other than the s-st. ones, additional tests are performed on the reasoning results: For every bridge rule set to 1 in the BRM, a test is performed whether one of its conditions contains a "*not*"-statement. If this test yields a positive result, the found equilibrium is not an s-st. equilibrium, as its status as equilibrium is based on the absence of the knowledge base element contained in the "*not*"-condition. Then, a test for unfoundedly inactivated 0-bridge rules in the BRM has to be performed analogously as in "$findEquilibria$".

Finally, we add the content of the knowledge bases initially handed over when invoking "$findSstEquilibria$" to the belief state obtained as equilibrium, and perform another reasoner call for every context, assuring the consistency of the expanded belief state. If all reasoner calls yield a valid result, the equilibrium found is a s-st. equilibrium of the MCS. The BRM and the expanded belief state are linked and jointly added to the list of s-st. equilibria. Then, "$findSstEquilibria$" proceeds with the next BRM (if there is any).

4.4 Comments Concerning Algorithmic Complexity

As a measure for the complexity of the "$findEquilibria$" routine we may use the number of reasoner calls performed. Given an MCS with m contexts and n BRMs. In the worst case, at least $m \cdot n$ reasoner calls have to be performed. Therefore, in order to identify all equilibria of an MCS, in the worst case at least $m \cdot 2^k$ reasoner calls (k the number of bridge rules in the MCS) are needed.[9]

In real world applications, when all equilibria of an MCS ought to be found, the dominating element in this estimation is the number of bridge rules k. For the naive generate and test approach, in the most general case the number of $m \cdot 2^k$ reasoner calls is a hard lower boundary. All BRMs have to be tested, and in every test for every context a reasoner call has to be performed.

[8] We call an equilibrium self-sustaining iff the application of all bridge rules may only be based on the application of other bridge rules, and thus be independent of the concrete knowledge bases of the MCS.

[9] Always assuming that per context per BRM one reasoner call is in fact needed and is sufficient. Here, sufficiency may be assured for the different types of contexts of finite generalized MCS (as admissible for the algorithm presented in [4]).

For the "$findSstEquilibria$" mechanism, in the most general case $m \cdot (2^k - 1)$ is a hard lower boundary for the number of reasoner calls (m the number of contexts of the MCS, k the number of bridge rules). In the average case, the number of reasoner calls will be higher: For every BRM which in fact represents a s-st. equilibrium, m^2 reasoner calls have to be performed.

5 Conclusion

When applying the MCS framework to different testing examples, it has shown to be applicable and appropriate (see [7] and [8] for examples). For the future, more sophisticated techniques for reducing computational complexity will be needed. Nevertheless, the approach to implementing an MCS has proven to be effective, and an important step towards the implementation of an MCS framework has been made.

References

1. Roelofsen, F., Serafini, L.: Minimal and Absent Information in Contexts. In: International Joint Conference on Artificial Intelligence. Volume 19., Lawrence Erlbaum Associates LTD (2005) 558
2. Brewka, G., Roelofsen, F., Serafini, L.: Contextual Default Reasoning. Proc. IJCAI-07, Hyderabad, India (2007)
3. Brewka, G., Eiter, T.: Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In: Proceedings of the National Conference on Artificial Intelligence. Volume 22., Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2007) 385-390
4. Besold, T.R., Mandl, S.: Integrating Logical and Sub-Symbolic Contexts of Reasoning. In Filipa, J., Fred, A., Sharp, B., eds.: Proceedings of ICAART 2010 - Second International Conference on Agents and Artifcial Intelligence, INSTICC Press (2010) . For a full version of the paper see also http://www8.informatik.uni-erlangen.de/inf8/Publications/bridging_mcs_original.pdf.
5. Syrjnen, T.: Lparse 1.0 User's Manual. (2000)
6. Simons, P., Niemela, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. 138(1-2) (2002) 181-234
7. Besold, T.R., Schiemann, B.: A Multi-Context System Computing Modalities. In: Proc. 23rd Int.Workshop on Description Logics (DL2010),Waterloo, Canada. Number 573 in CEUR Workshop Proceedings (2010)
8. Besold, T.R.: Theory and Implementation of Multi-Context Systems Containing Logical and Sub-Symbolic Contexts of Reasoning. Master's thesis, Department of Mathematics & Department of Computer Science 8: Artificial Intelligence, FAU Erlangen-Nuremberg (2009) . The full thesis is available under http://www.opus.ub.uni-erlangen.de/opus/volltexte/2010/1587/.