Wellington 1.0 User Manual

Ulrich Endriss

Department of Computer Science, King's College London,
Strand, London WC2R 2LS, United Kingdom
Email: `endriss@dcs.kcl.ac.uk`

Abstract

We present an overview of the basic functionality of WELLINGTON 1.0, the first public release of the description logics based knowledge representation and reasoning system developed by the Group of Logic and Computation at King's College London. This paper also provides information on how to obtain the software and includes a very brief introduction to the field of description logics itself.

1 Introduction

WELLINGTON is a description logics based knowledge representation system that is currently being developed by the Group of Logic and Computation at King's College London. Version 1.0, introduced in [3], essentially implements an ABox reasoner for the standard description logic $\mathcal{ALC}$. The software is available from the WELLINGTON web site:

`http://www.dcs.kcl.ac.uk/research/groups/logic/wellington/`

At the time of writing, this site provides access to WELLINGTON 1.0 and gives a short overview of description logics related research in our group.

WELLINGTON 1.0 is freely available over the Internet. It has been implemented in Java and can be run either as an applet through a web browser or as a Java application. Section 2 tells you how. In Section 3 we review the basic notions of description logics as far as they are relevant for the functionality currently available in WELLINGTON. This includes a definition of the description logic $\mathcal{ALC}$. How formulas in this language are represented in WELLINGTON is explained in Section 4. Section 5 presents the graphical user interface of the software and briefly describes every function that may be executed by the user. This includes in particular the description logical reasoning services provided by WELLINGTON. We conclude with some remarks regarding the implementation of the current system.

Technical Report TR-01-01, Department of Computer Science, King's College London, March 2001.

2 Getting and Running Wellington

The program may be run as an applet directly from our web site (provided your browser can handle Java 1.2 applets). However, in order to be able to use the system's full functionality (in particular loading and writing files) the Java archive `wellington.jar` should be downloaded and Wellington should be run as an application. Like this:

```
java -classpath wellington.jar
      uk.ac.kcl.dcs.wellington.gui.MainApplication
```

If you are familiar with Java, you may prefer to add `wellington.jar` to your system's classpath in the first place.

The next session covers some theoretical background on description logics. Sections 4 and 5 describe the syntax of formulas Wellington can process and explain the various functions accessible through the graphical user interface.

3 Description Logics

Description logics are formal knowledge representation languages with a relatively simple syntax and well-defined semantics. According to the description logic paradigm, knowledge is divided into a terminological part (TBox), where concepts like *"movies that are comedies and have no actors who are famous"* and relations holding between such concepts are defined, and an assertional part (ABox), where individuals are related to each other and asserted as being instances of certain concepts.

The central notion of description logics is that of a *concept*. Concepts are sets of objects (often called *individuals*). Some of the concept-building operators, like for example conjunction, directly correspond to standard set operators (like intersection). On top of that, $\mathcal{ALC}$ offers two kinds of quantification operations. In description logics, quantification is restricted to objects that are related to some reference object via a given binary relation. These relations are called *roles*. For example, the concept $\forall r.C$ denotes the set of objects a for which *every* object b, that is related to a via the role r, belongs to the concept C.

Given a set of *concept names* and a set of *role names* the set of valid *concept formulas* of $\mathcal{ALC}$ may be defined inductively. Any concept name is also a concept formula. Let C and D be concept formulas and let r be a role name. Then also $\neg C$ (negation), $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\forall r.C$ (value restriction), and $\exists r.C$ (existential restriction) are valid concept formulas. So are $\top$ ("top") and $\bot$ ("bottom"). We may assign a concept formula the status of a *terminological axiom* to express that we want every individual to belong to that concept. A set of terminological axioms is called a *TBox*. Here's an example for a concept formula:

$$\mathit{Movie} \;\sqcap\; \mathit{Comedy} \;\sqcap\; \neg\,\exists \mathit{actor}.\mathit{Famous}$$

Negation	$\neg C$	$\Delta \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Value restriction	$\forall r.C$	$\{a \in \Delta \mid \{b \in \Delta \mid (a,b) \in r^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}\}$
Existential restriction	$\exists r.C$	$\{a \in \Delta \mid \{b \in \Delta \mid (a,b) \in r^{\mathcal{I}}\} \cap C^{\mathcal{I}} \neq \{\}\}$

Figure 1: Syntax and Semantics of $\mathcal{ALC}$ Concepts

The operations of implication and equivalence can be defined in terms of disjunction and negation in the usual way. We have $C \Rightarrow D \equiv \neg C \sqcup D$ and $C \Leftrightarrow D \equiv (C \Rightarrow D) \sqcap (D \Rightarrow C)$.

The semantics of a concept formula are defined in terms of a *domain* Δ and an *interpretation function* $\mathcal{I}$. Every concept name is interpreted as a subset of Δ. Every role name is interpreted as a subset of $\Delta \times \Delta$, the set of pairs over the domain. The interpretation of complex concept formulas is defined in Figure 1. An interpretation is a *model* for a terminological axiom C iff it satisfies $C^{\mathcal{I}} = \Delta$. The model of a TBox is an interpretation that is a model for every formula in that TBox.

In the context of knowledge representation applications, terminological axioms are often restricted to formulas of the form $C \dot{\sqsubseteq} D$ (short for $\neg C \sqcup D$) and $C \doteq D$ (short for $(\neg C \sqcup D) \sqcap (C \sqcup \neg D)$), where C is a concept name. Formulas of the latter kind are called concept definitions; those of the former kind are commonly referred to as primitive concept definitions. Observe that $(C \dot{\sqsubseteq} D)^{\mathcal{I}} = \Delta$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $(C \doteq D)^{\mathcal{I}} = \Delta$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$. That is, from a purely *logical* perspective $\dot{\sqsubseteq}$ is nothing but an implication and $\doteq$ coincides with $\Leftrightarrow$.

To express assertional knowledge we introduce a set of *individual names*. Let a and b be names of individuals, let r be a role name, and let C be a concept formula. We distinguish two kinds of assertions. A *relational assertion* is of the form $(a, b) : r$ and asserts a and b as being related via the role r. An *instantiational assertion* $a : C$ asserts a as belonging to C. A set of assertions is called an *ABox*. An ABox together with a TBox is called a *knowledge base*.

The interpretation function $\mathcal{I}$ maps individuals to elements of the domain Δ. A relational assertion $(a, b) : r$ is satisfied by an interpretation iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ and an instantiational assertion $a : C$ is satisfied iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$. An interpretation is called a *model* for an ABox, if it satisfies all the assertions in that ABox. It is called a model for an ABox with respect to a TBox, if it is a model for both of them.

Typical reasoning services include *concept subsumption*, *concept consistency*, *ABox consistency*, and *instance checking*. The former two only concern the terminological part of a description logical system. A concept C is said to be subsumed by another concept D iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for every interpretation. We also speak about subsumption with respect to a TBox, namely whenever that subset-relation holds for all models of a TBox. We may *classify* a TBox by computing the subsumption relation for every pair of atomic concepts mentioned in that TBox. A concept formula C is consistent (with respect to to a TBox) iff there is an interpretation (a model of the TBox) for which $C^{\mathcal{I}}$ is not the empty set. An ABox is consistent (with respect to a TBox) iff it has a model (with respect to that TBox). An individual a is an instance of a concept C (with respect to a TBox) iff for every interpretation (every model of that TBox) $a^{\mathcal{I}}$ is in the set $C^{\mathcal{I}}$. Observe that all these inference services can be reduced to ABox consistency checking (possibly with respect to a TBox). A concept C is consistent iff the ABox $\{a : C\}$ (for some individual name a) is consistent. Furthermore, C is subsumed by D iff the ABox $\{a : C \sqcap \neg D\}$ is inconsistent. Finally, we can infer that a needs to be an instance of C iff adding $a : \neg C$ to the ABox renders it inconsistent.

Reasoning with respect to a TBox is considerably more complex than pure ABox reasoning. If formulas in the TBox are all of the form $C \dot{\sqsubseteq} D$ or $C \doteq D$, where C is a concept name, and if these definitions are acyclic, we may replace every occurrence of such a defined concept C in the ABox with the respective concept formula through a process known as *unfolding* and make the TBox obsolete. In the general case, however, this might not be possible.

For further information on description logics we refer to [1] for a very readable introduction to the field, which includes an extensive bibliography. The next section documents how formulas in $\mathcal{ALC}$ are represented in the WELLINGTON system.

4 Syntax of the Input Language

In this section we define the syntax of the input language for $\mathcal{ALC}$ formulas used in WELLINGTON. This language complies to the quasi-standard set by the description logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort in 1993 [7].

In the future, we plan to extend WELLINGTON's reasoning engine to cover a description logic enriched with arithmetical constraints over role fillers and a number of other non-standard features. Such a logic has first been introduced in [6] and [5]. Wellington 1.0 can already be used to manage knowledge bases in that language, but implementations of the reasoning services have not yet been included in the current release. Therefore, the following specification is restricted to the syntax of formulas in $\mathcal{ALC}$.

4.1 Concept Formulas

We start by describing the grammar of *concept formulas* in Backus-Naur-form (BNF). Expressions starting with a capital letter are non-terminals. Lowercase expressions are terminals, i.e. these are typed into the system as they appear here. With `STRING` we mean arbitrary alphanumeric strings (plus underscore _ and hyphen `-`) that start with a letter.

Note that in BNF a bar `|` denotes a choice. A `ConceptFormula`, for example, could be either a `PropositionalAtom`, or a `ConceptNegation`, or etc. A plus `+` after an expression denotes a non-empty list of expressions of the kind described.

```
ConceptFormula      ::= PropositionalAtom |
                        ConceptNegation |
                        ConceptConjunction |
                        ConceptDisjunction |
                        ConceptImplication |
                        ConceptEquivalence |
                        QuantifiedFormula |
                        (ConceptFormula)

PropositionalAtom   ::= ConceptName |
                        top |
                        bottom

ConceptName         ::= STRING

ConceptNegation     ::= (not ConceptFormula)

ConceptConjunction  ::= (and ConceptFormula+)

ConceptDisjunction  ::= (or ConceptFormula+)

ConceptImplication  ::= (implies ConceptFormula ConceptFormula)

ConceptEquivalence  ::= (equivalent ConceptFormula ConceptFormula)

QuantifiedFormula   ::= (all Role ConceptFormula) |
                        (some Role ConceptFormula)

Role                ::= STRING
```

Consider, for example, the following concept expression describing the set of all movies that aren't comedies and that have an actor who is of the "hero type":

$$Movie \sqcap \neg Comedy \sqcap \exists hasActor.HeroType$$

In WELLINGTON syntax, this formula would be written as follows:

```
(and Movie (not Comedy) (some hasActor HeroType))
```

In this example a few additional pairs of parentheses won't matter. This is because of the last line in the definition of `ConceptFormula`.

The atom `top` (or $\top$) denotes the universal concept to which every object belongs and which is a superconcept to any given concept formula. Analogously, `bottom` (or $\bot$) stands for the inconsistent (or empty) concept to which no object belongs and which is a subconcepts to any given concept formula.

As pointed out in the section on description logics, a concept implication like $C \Rightarrow D$ is logically equivalent to the primitive concept definition $C \dot{\sqsubseteq} D$. It is part of the WELLINGTON philosophy not to assign concept definitions any highlighted status. Moreover, concept names need not be defined as such; they can simply be used within formulas. To comply with the syntax of other systems, however, a number of synonymous keywords have been defined. For example, you may use `define-concept` instead of `equivalent`, or `define-primitive-concept` instead of `implies`.

4.2 ABox Formulas

The assertional part of a knowledge base is a list of *ABox formulas*. A major component of those are the previously introduced concept formulas. An ABox formula is either of *instantiational* or of *relational* type. Here's the grammar in BNF:

```
ABoxFormula    ::= (instance ABoxIndividual ConceptFormula) |
                   (related ABoxIndividual ABoxIndividual Role)

ABoxIndividual ::= STRING
```

As an example, consider the following little ABox, which contains two assertions, an instantiational and a relational one:

$$\begin{array}{rcl} \mathit{juliaRoberts} & : & \mathit{Actress} \sqcap \mathit{Famous} \\ (\mathit{juliaRoberts}, \mathit{prettyWoman}) & : & \mathit{actsIn} \end{array}$$

Here, *juliaRoberts* and *prettyWoman* are ABox individuals, $\mathit{Actress} \sqcap \mathit{Famous}$ is a concept formula, and *actsIn* is a role. In WELLINGTON we would encode this as follows:

```
(instance juliaRoberts (and Actress Famous))
(related juliaRoberts prettyWoman actsIn)
```

We use the same kind of strings for concept names, roles, and ABox individuals. Only by the structure of the formulas submitted to the system WELLINGTON is able to determine what category a given term belongs to. In particular, you could use the same string, say `julia`, to denote an individual, a role, and a concept name within the same knowledge base.

4.3 Pattern Matching

WELLINGTON is equipped with a simple *search function* to find formulas of interest in a large knowledge base. You may search for specific formulas or you may search for groups of formulas at a time by providing a *pattern* describing the kind of formulas you are interested in. WELLINGTON's pattern matching mechanism has been inspired by the anonymous variable construct used in the Prolog programming language. The underscore _ may represent *any* syntactically valid subformula of a TBox or an ABox formula (this includes role and individual names).

For example, in order to find all instantiational assertions about the individual `juliaRoberts` that are present in the current knowledge base we could search for the following pattern:

```
(instance juliaRoberts _)
```

As another example, suppose we were interested (for one reason or the other) in all concept formulas in the TBox that are implications whose consequent is an existentially quantified concept with an unspecified role referring to the concept `VeryLarge`. The corresponding pattern would be:

```
(implies _ (some _ VeryLarge))
```

The *delete function* uses the same pattern matching mechanism. For the above example all formulas matching the query would be deleted from the current knowledge base.

5 User Interface and Reasoning Services

The WELLINGTON interface consists of a (small) text field for user input (description logical formulas and patterns), a larger text area on which WELLINGTON will print any output, four buttons, and a number of menus. This is shown in Figure 2. Some of the menu options will also launch some very simple data input dialogues.

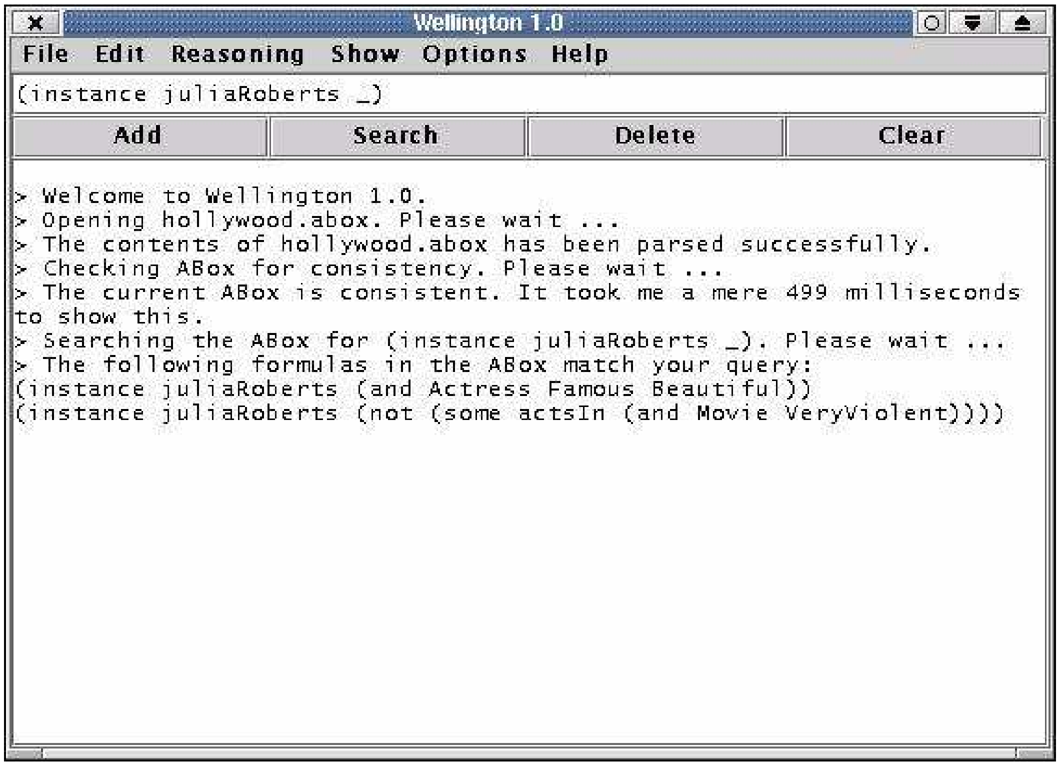
We separate the description of each menu option and push button into issues relating to the *management of knowledge bases* (mainly buttons, *File*, and *Show* menu) and *reasoning services* (*Reasoning* and *Options* menu), respectively.

5.1 Managing Knowledge Bases

File menu. Choose the *Open file* option to open a file containing a list of TBox and ABox formulas specified in Ohlbach's description logic with arithmetical constraints [5] (which includes $\mathcal{ALC}$). The contents will be parsed automatically and added to the current knowledge base. If there are any syntax errors the first error is reported to the user and no changes are made to the current knowledge base.

The knowledge base currently in memory can be written to a file by selecting the *Save KB as* option. *Reset* will clear the entire knowledge base. The *File* menu is also used to exit WELLINGTON.

Show menu. The *Show* menu can be used to view the currently loaded ABox or TBox, respectively. The TBox is separated into *concept formulas* and *role axioms.* Please note that there are no role axioms in standard $\mathcal{ALC}$.



```
(instance juliaRoberts _)
```

```
> Welcome to Wellington 1.0.
> Opening hollywood.abox. Please wait ...
> The contents of hollywood.abox has been parsed successfully.
> Checking ABox for consistency. Please wait ...
> The current ABox is consistent. It took me a mere 499 milliseconds
to show this.
> Searching the ABox for (instance juliaRoberts _). Please wait ...
> The following formulas in the ABox match your query:
(instance juliaRoberts (and Actress Famous Beautiful))
(instance juliaRoberts (not (some actsIn (and Movie VeryViolent))))
```

Figure 2: The Wellington Interface

Adding, searching, and deleting formulas. To add a single ABox or TBox formula, type it into the input text field and press the *Add* button. Syntax errors will be reported where applicable. To search for a formula or a pattern in the current knowledge base, type it into the input field and press *Search*. Pressing the *Delete* button will delete all formulas from the current knowledge base that match the pattern given in the input field.

5.2 Reasoning Services

Reasoning menu. Choosing the *ABox consistency* option will check the current ABox for consistency. Note that reasoning with respect to a TBox has not yet been implemented, so TBox formulas in the knowledge base will simply be ignored. Both the *Concept consistency* and the *Concept subsumption* options will result in a dialogue window being launched to put in the concept formula(s) in question. Please recall that these two reasoning services are independent of the currently defined knowledge base.

So far, reasoning is restricted to ABoxes and concept formulas in $\mathcal{ALC}$. If any of the formulas involved are not in $\mathcal{ALC}$, an error message will be issued.

Recall that you can perform an instance check for an individual a and a concept formula C (with respect to the currently defined ABox) by adding $a : \neg C$ to the knowledge base and checking the resulting ABox for inconsistency.

Options menu. At this stage there's just one option available. You may change the *timeout* value, that is the time (in milliseconds) after which any inference process should be interrupted.

5.3 Other Items

The *Edit* menu provides cut and paste facilities. Pushing the *Clear* button will clear the input text field. Use the *Edit* menu to clear the output text area. The *Help* function has not been implemented yet.

6 Conclusion

We have presented the description logics based knowledge representation system WELLINGTON. It currently implements a Tableaux-like calculus to check the consistency of the assertional component of a knowledge base specified in $\mathcal{ALC}$. Other reasoning services are based on this core algorithm. It incorporates a number of well-known optimisation techniques, including lexical normalisation, semantic branching with heuristic guided search, beta simplification, boolean constraint propagation, and backjumping. We refer to [2] for a recent survey paper on Tableaux for description logics and to [4] for a good overview of optimisation techniques for these calculi.

Acknowledgements. The work described in this paper is part of the Data Driven Logic Algorithms project at King's College and has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant reference number GR/L91818. Thanks to Stefan Schlobach and Hans Jürgen Ohlbach for many helpful discussions. WELLINGTON wouldn't be what it is today ...

References

[1] Franz Baader. Logic-based knowledge representation. In M. J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*. Springer-Verlag, 1999.

[2] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Automated Reasoning with Tableaux and Related Methods, Proceedings of Tableaux'2000*, number 1847 in LNAI, pages 1–18. Springer-Verlag, 2000.

[3] Ulrich Endriss. Reasoning in description logics with Wellington 1.0 – system description. In H. J. Ohlbach, U. Endriss, O. Rodrigues, and S. Schlobach, editors, *Proceedings of the Seventh Workshop on Automated Reasoning, Bridging the Gap between Theory and Practice*, volume 32 of *CEUR Workshop Proceedings*, July 2000.

[4] Ian Horrocks and Peter F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[5] Hans Jürgen Ohlbach. A theory resolution style ABox calculus. Extended abstract. In *M4M, Methods for Modalities 1, Workshop Proceedings.* ILLC, University of Amsterdam, 1999.

[6] Hans Jürgen Ohlbach and Jana Koehler. Modal logics, description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1–2):1–31, 1999.

[7] Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort, November 1993.