Logic and Constraints Assignment – CS-E4800 Artificial Intelligence

Jussi Rintanen
Aalto University
Department of Computer Science

February 13, 2017

Abstract

This document describes the modeling language used in the Artificial Intelligence course. The language is essentially propositional logic extended with schematic definitions, including simple integer arithmetics to refer to different atomic propositions. The language is reduced to the SMTLIB format to be fed to SMT solver (limited to Boolean variables.)

Contents

Foreword

Logic is a core part of much of advanced software technologies in artificial intelligence, formal methods, programming languages, and databases. Most of the best solvers for the most important logic, the propositional logic, take their input in very-low-level formats, such as the DIMACS CNF format (most solvers for the satisfiability problem SAT), or the SMTLIB format (Satisfiability modulo Theories solvers).

For educational use these formats are way too low level, when used for exercises and assignments that should be possible to complete in a couple of hours.

For this reason, we have used a higher level language that includes schematic definitions of propositional formulas and important cardinality constraints that show up in many applications, bringing the language closer to the predicate logic, but preserving the possibility to easily reduce it to the input formats of the best SAT solvers.

1 Language Definition

1.1 Lexical Entities

The language is case-sensitive.

Symbols are alphanumeric objects, that is, strings that start with a letter and are followed by zero or more letters or numbers.

Comments are any text that starts with // and continues until the end of a line.

1.2 Atomic Propositions

Atomic propositions consist of

1. *predicate* symbol,
2. a left parenthesis
3. a comma-delimited list of *terms*,
4. a right parenthesis.

A *term* is

1. *symbol*
2. an integer constant ≥ 0
3. *term* + *term*
4. *term* - *term*
5. *term* * *term*

All terms that contain the arithmetic operations must consist of only two types of atomic terms, integer constants or integer variables bound by *quantifiers* (see Section 1.4).

All arithmetic terms are evaluated *during the grounding process*, and integers only appear as parts of *names* of atomic propositions in the resulting SMTLIB specifications.

Any atomic propositions can be used: predicate symbols or atomic terms do not need to be declared in advance in any way.

1.3 Formulas

A *formula* is

1. an *atomic proposition*
2. *formula* & *formula*
3. *formula* | *formula*
4. **not** *formula*
5. *formula* −> *formula*
6. *formula* <−> *formula*
7. *formula* **implies** *formula*
8. **forsome** *symbol* : *setexpr formula* **end**

9. **forall** *symbol* : *setexpr formula* **end**
10. **exactlyone** *symbol* : *setexpr formula* **end**
11. **atmostone** *symbol* : *setexpr formula* **end**
12. (*formula*)
13. *term* = *term*
14. *term* != *term*
15. *term* > *term*
16. *term* < *term*
17. *term* >= *term*
18. *term* <= *term*

The terms in the last four inequalities must be numeric terms.

Parentheses are used for reducing ambiguity. *There is no default precedence for the connectives.* For example, $A\&B <- > C$ must be clarified with parentheses to either $(A\&B) <- > C$ or $A\&(B <- > C)$.

Above, *setexpr* is a reference to a *set* of *terms*, expressed as

1. named *type* (see Section 2.1.)
2. integer interval $[l..u]$ where l and u are the lower and upper bound of an integer range
3. set of symbols expressed as { symbol$_1$, symbol$_2$, ..., symbol$_n$ }

1.4 Quantifiers

Quantifiers *forsome*, *forall*, are reduced to disjunctions and conjunctions, respectively. Quantifiers *atmostone* is reduced to formulas $not(\phi(i)\&\phi(j))$ with all possible instantiations of i and j (with $i \neq j$) from the variable bound with the associated setexpr. The quanfier *exactlyone* is a combination of *forsome* and *atmostone*.

The *variable* quantified by the quantifier is the symbol occurring right after the quantifier keyword. It obtains all possible values from the associated *setexpr*.

2 Structure of the Specification

2.1 Named Types

Sets of terms can be named with *type* declarations.

```
type locations = { Helsinki, Vantaa, Espoo, Tampere, Turku, Rovaniemi };
type HOUSE = { 1,2,3,4,5 };
```

2.2 Named Constants

Integer values can be named for later use.

```
let horizon = 10;
```

Named constants can be used as a part of an arithmetic expression or as a part of an integer range expression.

2.3 The Formulas

The rest of the specification consists of one or more formulas. Like all other declarations, formulas must end in a semicolon.

```
forsome i : PERSON j : PERSON k : HOUSE l : HOUSE
   (Smokes(i,Chesterfields) &
    Pet(j,Fox) &
    LivesIn(i,k) &
    LivesIn(j,l) &
    (k = l+1 | l = k+1))
end;
```