ASPeRiX
Quick start manual

Claire Lefèvre,
Pascal Nicolas,
Stéphane Ngoma,
Christopher Béatrix

April 18, 2013

Contents

1 Brief description

ASPeRiX is an ASP solver developed in C++. It is an implementation of the stable model semantics for normal logic programs. The main specifity of this system is to realize a forward chaining of first order rules that are grounded on the fly. So, unlike others available ASP systems, ASPeRiX does not need a pregrounding processing.

2 Usage

Answer set(s) of a program prog is computed as follows :

$ *asperix n prog*

where n is the (maximum) number of answer sets to be computed (zero indicates all). If n is missing, only one answer set is computed.

Option -N k fixes the interval of integer constant accepted as $[-k..k]$ (1024 by default).
Option -F k fixes k as the maximum nesting level accepted for functional terms (16 by default).
Option -*verbose* prints extra information about the computation.
Option --*help* displays help information of ASPeRiX.

3 Language features

ASPeRiX is able to deal directly with any normal logic program containing rules with variables, function symbols, lists and arithmetic calculus. The only syntactic restriction is the safety of rules, i.e. all variables occurring in the rule must also occur in its positive body (cf 3.1).

3.1 Rule

ASPeRiX allows 3 types of rules:

- Fact rule: *head*.
- Normal rule: *head* :- *body*.
- Constraint rule: :- *body*.

where *head* is a predicate atom and *body* a list of literals and NAF-literals (Negation As Failure-literals) separated by commas.

Examples :

- $p(1)$.

- $q(Y+1)$:- $p(Y)$, Y != 3, *not* $q(Y)$.
- :- -$q(2)$.

Safety of rule

A rule is safe if each variable that appears in the rule appears also in the positive body of the rule either in a predicate atom or on the left side of an assignment atom.

Examples :

- $a(X)$:- $b(X)$, *not* c. is a safe rule.
- $a(X)$:- $X = 2$, *not* b. is a safe rule.
- $a(X)$:- $b(X)$, *not* $c(Y)$. is an unsafe rule.

3.2 Term

A term can be a numeric constant, a symbolic constant, a string constant, a functional term, a list term, a variable, a built-in predicate or an arithmetic expression.

3.2.1 Numeric constant

A numeric constant is an integer in $[-k..k]$ interval. Maximum integer constant is given by -N option (1024 by default).

Examples : 0, 512, -28, ...

3.2.2 Symbolic constant

A symbolic constant is a string of letters, digits and underscores starting with a lower case letter.

Examples : *toto*, *a*1*b*2, *a_B*, ...

3.2.3 String constant

A string constant is a sequence of characters enclosed within double quotes.

Examples : "*toto*", "*XyZ*", "1*Ab*", "**z*2-*", ...

3.2.4 Functional term

A functional term is a functor (symbolic constant or string constant) followed by a parenthesized list of terms. Maximum nesting level is given by -F option (16 by default).

Examples : $f(1,2,3)$, $f2(a,2+X)$, "f"$(g(h([1,2])))$, ...

3.2.5 List term

A list term has form $[t_1,...,t_n]$ where $t_1,...,t_n$ are terms or $[t|l]$ where t is a term and l is a list term.

Examples : [], $[a,b,c]$, $[a|[b,c]]$, $[1,f(X),7*X]$, ...

3.2.6 Variable

A variable is a string of letters, digits and underscores starting with an upper case letter.

Examples : X, X_2, X_y_Z, ...

3.2.7 Built-in predicate

A built-in predicate is a predefined predicate starting with a # (see 3.6).

Examples : $\#member(c,[a,b,c])$, $\#head([a,b,c],a)$, ...

3.2.8 Arithmetic expression

An arithmetic expression is built with numeric constants, variables, built-in predicate, binary operators $+,-,*,/$, modulo operator *mod*, unary operator *abs* (absolute value of an arithmetic expression), and parenthesis. An arithmetic expression must be grounded when evaluated.

Examples : $4*3$, $(\#length([a,b,c])-2)$, $(6\ mod\ 2)$, $X+3$, ...

3.3 Atom

An atom is either a predicate atom, a relational atom or an assignment atom.

3.3.1 Predicate atom

A predicate atom is a predicate symbol (symbolic constant or string constant), optionally followed by a parenthesized list of terms.

Examples : $toto$, $f(X)$, $f(g(titi, 2))$, ...

Additionally, a rule head can be a predicate symbol followed by a parenthesized list of range terms $t_1..t_2$ where t_1 and t_2 are numeric constants or variables.

Examples : A fact $n(1..5)$. is equivalent to $n(1)$. $n(2)$. $n(3)$. $n(4)$. $n(5)$.

3.3.2 Relational atom

A relational atom is a comparison between two terms. Operators are ==, !=, <, <=, >, >=. A relational atom must be grounded when evaluated. Relational atoms can only occur in the positive body of a rule.

Examples : $4 >= -7$, $toto$!= 2, $f(a, b, c) < [1, 2]$, $\#length([a, b, c]) > X$, ...

There is a fixed ordering over all terms as defined in the current version of ASP-Core-2 Input Language Format[1]: numeric constant $\preceq$ symbolic constant $\preceq$ string constant $\preceq$ functional term $\preceq$ list term. A numeric constant $a \preceq b$ iff $a \leq b$. Symbolic constants and string constant are lexicographically ordered. Functional terms are ordered firstly by their arities, secondly by their names, and finally by their arguments. List terms are ordered by their arities and then by their arguments.

Examples (numeric terms) : $2 \preceq 3$, $23 \preceq toto$ $56 \preceq f(a)$ $42 \preceq [a, b]$ ...

Examples (symbolic and string terms) : $titi \preceq toto$ $toto \preceq$ "$titi$" "$titi$" $\preceq$ "$toto$" $titi \preceq f(a)$ $titi \preceq [1, 2]$ ...

Examples (functional terms) : $q(a, b) \preceq p(a, b, c)$ $p(a, b) \preceq q(a, b)$ $p(a, b) \preceq p(b, b)$ $p(a, b, c) \preceq [1, 2]$ ...

Examples (list terms) : $[a, b] \preceq [1, 2, 3]$ $[a, b] \preceq [b, c]$ ...

3.3.3 Assignment atom

An assignment atom has form $variable = term$. Assignment atoms can only occur in the positive body of a rule.

Examples : $X = 2$, $Y = f(X)$, $Z = \#head([a, b, c])$, ...

[1] https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf

3.4 Literal

A literal is an atom a or strong negation of a predicate atom -a. Also, NAF-literal *not* a is allowed where a is a predicate atom or the negation of a predicate atom.

Examples : -*toto*, *not* $f(a)$, *not* -*titi*, ...

3.5 Comments

A comment line begin with a "%" and ends at the end of the line.

Examples : *%This is a comment!*

3.6 Lists

External predicates are available to manipulate lists. The dynamic library *lists.so* must be loaded in order to use them (cf 4.2 to see the command). This library is available in the current version of `ASPeRiX` in the same directory than the executable. The symbol "+" before an argument represents an input parameter whereas the symbol "-" indicates an output parameter.

- $\#append(+List1, +List2, -List3)$ is true if $List3$ is the concatenation of the lists $List1$ and $List2$.
- $\#del(+List1, +Term, -List2)$ is true if $List2$ is the list $List1$ without all occurrences of the element $Term$.
- $\#delFirst(+List1, +Term, -List2)$ is true if $List2$ is the list $List1$ without the first occurrence of the element $Term$.
- $\#delNth(+List1, +N, -List2)$ is true if $List2$ is the list $List1$ without the element at position N.
- $\#head(+List, -Term)$ is true if $Term$ is the first element of the list $List$.
- $\#insLast(+List1, +Term, -List2)$ is true if $List2$ is the list obtained by adding the element $Term$ to the end of the list $List1$.
- $\#insNth(+List1, +Term, +N, -List2)$ is true if $List2$ is the list obtained by adding the element $Term$ to the list $List1$ at position N.
- $\#last(+List, -Term)$ is true if $Term$ is the last element of the list $List$.
- $\#length(+List, -Size)$ is true if $Size$ is the length of the list $List$.
- $\#list(-List)$ is true if $List$ is the empty list.

- $\#list(+Term_1, ..., +Term_n, -List)$ $(n > 0)$ is true if $List$ is the list of the elements $Term_1, ..., Term_n$ in the same order.
- $\#member(+Term, +List)$ is true if the list $List$ contains the element $Term$.
- $\#memberNth(+List, +N, -Term)$ is true if $Term$ is the element at position N of the list $List$.
- $\#range(+Begin, +End, +Step, -List)$ is true if $List$ is the list of integers between $Begin$ and End with an interval of $Step$.
- $\#reverse(+List1, -List2)$ is true if $List2$ is the reversed list of $List1$;
- $\#subList(+List1, +List2)$ is true if $List1$ is a sublist of $List2$.
- $\#tail(+List1, -List2)$ is true if $List2$ is the list $List1$ without its first element.

Examples : $\#last([a, b, c], c)$, $\#reverse([a, b, c], X)$, ...

In addition, an external predicate can appear without its output parameter in a literal. In that case, the external predicate is substituted by the value of the output parameter during its computation.

Examples : $\#length([a, b, c]) > 2$, $f(\#tail([a, b, c]))$, $X = \#del([a, b, b, c], b)$, ...

4 Directives

4.1 Show / hide

These directives allow to hide some predicates when printing answer sets of a program. By default, all predicates are shown and a predicate named p with arity k can be hidden by *#hide p/k.*. The directive *#hide.* without arguments hides all predicates, and the *#show p/k.* directive can then be used to tell what atoms have to be printed.

Examples :

- *% Show only the predicate toto with arity 2 of a program*
  #hide. % Hide all predicates
  #show toto/2. % Show the predicate toto with arity 2
- *% Hide only the predicate titi with arity 3 of a program*
  #hide titi/3.

4.2 Include

This directive allow to load a dynamic library to use external predicates with the command *#include name* where *name* is a dynamic library surrounded by double quotes.

Example :
% Load the dynamic library lists.so
#include "lists"