XPCE User Manual

Shahin Saadati, Sam Owre, Shankar Natarajan, and Bruno Dutertre

June 4, 2009

1 Introduction

XPCE is based on MCSAT, and similar to PCE, but it uses XML-RPC and JSON in place of OAA and ICL.

2 Running XPCE

xpce PORT

This sets up an xpce server on the local host that listens to the given PORT (normally an unused port > 1024).

3 XPCE Methods

XPCE responds to a number of XML-RPC methods. In general, the methods expect JSON input strings and returns JSON strings. For the most part, the methods correspond to the commands of MCSAT. The return is generally a JSON object (i.e., enclosed in '{}'), and if there is a warning or error, it is included in the object as `"warning":` or `"error":` followed by a descriptive string.

3.1 `xpce.sort` - add a sort/supersort

- Argument: `{"name": NAME, "super": NAME}`

  `"super"` is optional

- Returns: `{}`
- Warnings:
- Errors:
  - `NAME is invalid`
  - `NAME is in use as a sort` - if `"super"` is not provided
  - `NAME is already a subsort of NAME`
- Description: If `"super"` is not provided, introduces new sort with the name `NAME`. Otherwise introduces new sorts as needed, and creates the subsort relation.

3.2 `xpce.predicate` - add a predicate

- Argument: `{"predicate": NAME, "arguments": NAMES, "observable": BOOL}`
- Returns: `{}`
- Warnings:
- Errors:
  - `NAME is invalid`
  - `NAME is in use as a predicate`
- Description: adds the NAME as a predicate, with signature given by the list of sorts in NAMES.

3.3 `xpce.const` - add constants of a given sort

- Argument: `{"names": NAMES, "sort": NAME}`
- Returns: `{}`
- Warnings:
- Errors:

  - `NAME is invalid`
  - `NAME is in use as a constant`

- Description: adds the NAMES as new constants of the given sort.

3.4 `xpce.assert` - assert a fact

- Argument: `{"fact": FACT}`
- Returns: `{}`
- Warnings:
- Errors:
- Description: asserts the `FACT` to the internal database. Note that facts are of the form `p(c1,...,cn)`, where `p` is an observable predicate, and the `ci` are all constants. Use `xpce.add` with a high weight for any other formulas.

3.5 `xpce.add` - add a weighted assertion or rule

- Argument: `{"formula": FORMULA, "weight": NUM, "source": NAME}`
  - `"weight"` is optional, defaults to `DBL_MAX`.
  - `"source"` is optional, no default
- Returns: `{}`
- Warnings:
- Errors:
- Description: asserts the `FORMULA` to the internal database with the given weight. The FORMULA may contain variables, which are instantiated with constants of the corresponding sort.

3.6 `xpce.ask` - query for instances

- Argument: `{"formula": FORMULA, "threshold": NUM, "maxresults": NUM}`
  - `"threshold"` is optional between 0.0 and 1.0 - default 0.0
  - `"maxresults"` is optional a nonnegative integer, default 0
- Returns: a JSON array of the form

```
[{"subst": SUBST, "formula_instance": FORMULA, "probability": NUM}
 ...
]
```

- Warnings:
- Errors:
- Creates instances of the given formula, runs MCSAT sampling, and collects the results, sorted according to probability (highest to lowest). Only returns results greater than or equal to the threshold, and at most maxresults are returned - unless it is zero, in which case all instances above the threshold are returned.

3.7 `xpce.command` - run an MCSAT command

This is the simplest, and the only one that does not expect a JSON string. It simply takes any of the commands as described in the MCSAT user guide in the form of a string (including the terminating ';'), and returns a string.

3.8 Nonterminals

```
FORMULA := ATOM
         | {"not": FORMULA}
         | {"and": [FORMULA, FORMULA]}
         | {"or": [FORMULA, FORMULA]}
         | {"implies": [FORMULA, FORMULA]}
         | {"iff": [FORMULA, FORMULA]}
```

```
ATOM := {"atom": {"predicate": NAME, "arguments": ARGUMENTS}}

NAMES := [NAME++',']
ARGUMENTS := [ARGUMENT++',']
CONSTANTS := [CONSTANT++',']
NUM := ['+'|'-'] simple floating point number
NAME := chars except whitespace parens ':' ',' ';'
ARGUMENT := CONSTANT | {"var":  NAME}
CONSTANT := NAME
```