Spatial Logic Model Checker User's Guide
version 0.6 (draft)

Hugo Vieira and Luís Caires

Departamento de Informática, FCT/UNL

December 2003

1 Introduction

Spatial logics support the specification not only of behavioral properties but also of structural properties of concurrent systems, in a fairly integrated way. Spatial properties arise naturally in the specification of distributed systems. In fact, many interesting properties of distributed systems are inherently spatial, for instance connectivity, stating that there is always an access route between two different sites, unique handling, stating that there is at most one server process listening on a given channel name, or resource availability, stating that a bound exists on the number of channels that can be allocated at a given location. Secrecy can also be sometimes understood in spatial terms, since a secret is a piece of data whose knowledge of is restricted to some parts of a system, and unforgeable by other parts.

The Spatial Logic Model Checker is a tool allowing the user to automatically verify behavioral and spatial properties of distributed and concurrent systems expressed in a pi-calculus. The algorithm implemented (currently using on-the-fly model-checking techniques) is provably correct for all processes, and complete for the class of bounded processes [1], an abstract class of processes that includes the finite control fragment of the pi-calculus. The tool itself is written in OCAML, and runs on any platform supported by the OCAML distribution.

For background on spatial logics for concurrency, see [4, 2, 3, 1] and other references therein. Forthcoming releases of this manual will include a short tutorial on the subject, some examples on how to use the tool, and a presentation of the underlying algorithms.

In this report we specify the syntax of the version of the pi-calculus currently supported in the tool, which is the synchronous polyadic pi-calculus, and the syntax of the spatial logic considered, which currently is in essence the logic described in [1], a spatial logic with behavioral and spatial operators and recursive formulas.

2 Syntax of Processes

Pi-calculus processes are specified according to the concrete syntax definition in Figure 1.

Understanding our syntax for the standard polyadic pi-calculus operators is straightforward. Note that restriction allows for the declaration of more than one restricted name in a row.

We adopt a CSP-like notation for input/output, so that in our syntax an output prefix $\overline{x}y_1y_2\ldots y_n$ is written $\texttt{x!(y}_1\texttt{,y}_2\texttt{,}\ldots\texttt{,y}_n\texttt{)}$, and an input prefix $x(y_1, y_2, \ldots, y_n)$ is written $\texttt{x?(y}_1\texttt{,y}_2\texttt{,}\ldots\texttt{,y}_n\texttt{)}$. Priority of process operators is defined as usual (restriction is more binding that composition), so that *e.g* the process

lower	::=	['*a*' – '*z*']
upper	::=	['*A*' – '*Z*']
letter	::=	*lower* | *upper*
digit	::=	['*0*' – '*9*']
name	::=	*lower* (*letter* | *digit* | '_')*
namelist	::=	ϵ | *name* (',' *name*)*
process	::=	0
	|	*process* | *process*
	|	`new` *namelist* `in` *process*
	|	*Id*(*namelist*)
	|	*name*`!`(*namelist*)`.` *process*
	|	*name*`?`(*namelist*)`.` *process*
	|	(*process*)

Figure 1: Syntax of Processes.

```
new secret in hand!<secret>.0 | erase?(x).0
```

is parsed as

```
(new secret in hand!<secret>.0) | erase?(x).0
```

rather than as

```
new secret in (hand!<secret>.0 | erase?(x).0)
```

The form *Id*(*namelist*) refers to a process defined using the `defproc` command in the toplevel command interpreter of the model-checker. This command, described below, allows the definition of sets of mutually recursive parametric processes.

3 Syntax of Formulas

Formulas of the spatial logic are specified according to the concrete syntax definition presented in Figure 2. Several of the connectives available are not primitive from a logical viewpoint, but have been directly implemented for the sake of efficiency.

The boolean connectives are negation `not`, conjunction `and`, disjunction `or`, implication `=>`, and equivalence (bi-implication) `<=>`.

Spatial connectives are void `void`, composition (or separation) `|`, decomposition `||` (de Morgan dual of composition), and revelation `reveal` (usually written ® [5]). We also include as a primitive connective the de Morgan dual of revelation `allreveal`, and the occurrence connective `free`.

Names can be tested for equality and inequality by the `==` and `!=` operators.

We then have quantifiers over names; the universal quantifier `forall`, the existential quantifier `exists`, the freshness quantifier `fresh`, and the hidden name quantifier `hidden`.

Behavioral modalities are `<`*label*`>`, expressing possibility of action (*cf.*, diamond modality of Hennessy-Milner logic), and its dual `[`*label*`]`, expressing necessity of action (*cf.*, the box modality of Hennessy-Milner logic. The argument *label* of the behavioral modalities specifies the (set of) actions considered. We have:

- `tau`, that denotes the silent action (an internal reduction step);
- *name*, that denotes any action (input or output) on subject *name*;
- `?`, that denotes any input action;
- `!`, that denotes any output action;
- *name*`!`, that denotes any output action on subject *name*;
- *name*`?`, that denotes any input action on subject *name*;
- *name*`?(`*namelist*`)`, that denotes a particular input action;
- *name*`!(`*namelist*`)`, that denotes a particular output action;
- `*`, that denotes any of the actions specified above.

Formulas can be introduced by (non recursive) parametric definitions, by a mechanism described below. Then $Id(idlist)$ denotes a defined property.

Last, but not least, properties can be defined by recursion, as in the mu-calculus and the spatial logics of [2, 1]: `mu` denotes the least fixpoint operator, and `nu` denotes the least fixpoint operator.

4 Running the Tool

After installation, the tool can be executed by issuing the command

```
% checker
```

in the operating system shell prompt. Currently, only a minimal command line interface is available.

5 Top level commands

In this section, we list the various commands that can be issued at the top level command prompt of the model checking tool.

```
formula  ::=  formula | formula
           |  formula || formula
           |  formula => formula
           |  formula <=> formula
           |  formula and formula
           |  formula or formula
           |  ( formula )
           |  not formula
           |  void
           |  true
           |  false
           |  name == name
           |  name != name
           |  Id(namelist)
           |  exists name . formula
           |  forall name . formula
           |  free name
           |  reveal name . formula
           |  allreveal name . formula
           |  hidden name . formula
           |  fresh name . formula
           |  <label>formula
           |  [label]formula
           |  mu Id.formula
           |  nu Id.formula
           |  Id(idlist)

label    ::=  tau
           |  name
           |  ?
           |  !
           |  name?
           |  name!
           |  name?(namelist)
           |  name!(namelist)
           |  *
```

Figure 2: Syntax of Formulas.

Process definition

`defproc` Id(*namelist*) $=$ *process* [`and` Id(*namelist*) $=$ *process*]* ;

Example

```
% defproc
  EchoServer(chan) =
    chan?(data,reply).(reply!(data).0 | EchoServer(chan))
and
  Client(chan) =
    new callback in
      (chan!(data,callback) | callback?(x).Client(chan))
and
  System() =
    new private in
      (Client(private) | EchoServer(private));
```

Property definition

`defprop` Id(*idlist*) $=$ *formula*;

Note that parameters of property identifiers can be either name parameters, or formula parameters.

Example

```
% defprop SImp(A,B)= not (A | not B);
```

Checking

`check` Id(*namelist*) `|=` *formula*;

Example

```
% check System() |= hidden x.SImp(<x!>true,[x!]false);
   yes.
```

Load

`load "`*`filename`*`";`
Executes the declarations and commands in the file whose pathname is obtained by the current path name by appending *`filename`*.

Change Path

`cd "`*`pathname`*`";`
Changes the current pathname to ***pathname***.

Show Path

`pd ;`
Shows the current pathname.

Clear

`clear;`
Clears the current session, erasing all process and formula definitions.

Trace

`trace` [`on` | `off`] `;`
Switches the trace level on or off.

List

`list` [`procs` | `props`]`;`
Lists the defined processes (`procs`) or properties (`props`).

Show

`show` *Id* `;`
Shows the process or formula assigned to the identifier *Id*.

Help

`help;`
Lists available commands.

Quit

`quit;`
Terminates the session.

Acknowledgements Work on the Spatial Logic Model Checker is funded by the UE project FET IST-2001-33100 Profundis.

References

[1] L. Caires. Behavioral and spatial properties in a logic for the pi-calculus. In Igor Walukiwicz, editor, *Proc. of Foundations of Software Science and Computation Structures'2004*, Lecture Notes in Computer Science. Springer Verlag, 2004.

[2] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186(2):194–235, 2003.

[3] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). *Theoretical Computer Science*, to appear.

[4] L. Cardelli and A. D. Gordon. Anytime, Anywhere. Modal Logics for Mobile Ambients. In *27th ACM Symp. on Principles of Programming Languages*, pages 365–377. ACM, 2000.

[5] L. Cardelli and A. D. Gordon. Logical Properties of Name Restriction. In S. Abramsky, editor, *Typed Lambda Calculi and Applications*, number 2044 in Lecture Notes in Computer Science. Springer-Verlag, 2001.