Syntactic Enhancements for IPDL

Kristina Sojakova Mihai Codescu

Acknowledgement

This project was funded through the NGI Assure Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No. 957073.

1 General Design

We have noticed the need to have two syntaxes for IPDL. So far, we have used the same representation internally and as input syntax for the user. These need to be separated in an abstract syntax (AS), used as internal representation, and a concrete syntax (CS), used as input syntax when writing actual IPDL case studies. The CS will be designed in the Milestone 7, together with the new language for proofs. The AS is what the rules of IPDL currently use, and it will be extended to accomodate new concepts or new notational variants, where need be. A parser of the CS to AS and a printer from AS to CS will be implemented in Milestone 7.

Remark 1 *Having the CS allows us to hide from the user many Maude-specific technicalities. For example, currently the user must write strategies instead of just calling them, and in the future they will be generated in the AS from a convenient CS notation.*

2 Subscripts and Superscripts

In the case studies, we often annotate channel names with subscripts and superscripts, with the meaning that the superscript denotes the sender and the subscript denotes the receiver. Maude does not provide syntactic support for this, so we need to introduce a convenient notation. This requires a refactoring of the datatype used for channel names. Channel names were so far of the form `a[ntlist]`, where `a` is an identifier and `nlist` is a list of identifiers, natural numbers or the result of applying several operations to these.

We have solved this by introducing a supersort of the Maude datatype of identifiers and a constructor for this supersort of the form `_^^_.._` taking three identifiers `a, b, c` as arguments and returning as result `a ^^ b .. c`, where `b` is the superscript and `c` is the subscript. Channel names have been extended to the form `a ^^ b .. c [nlist]`.

The examples will be updated to use the new notation.

Remark 2 *Maude requires spaces between the operators, so* `a^^b..c` *is invalid. The underline character is used in Maude for infix operators, so we use* `..` *instead.*

Remark 3 *In the CS we may use a notation closer to Latex.*

3 Compact Notation for Protocols

We need a representation of the protocols that is easier to process by a program, the main goal being interaction with a front-end.

We have introduced a notation that resembles JSON, although it does not follow it strictly. A transformation to JSON can be easily implemented, should it prove necessary. Of course, this is not currently in use in the case studies. Moreover, for debugging purposes it is useful to inspect the shape of a protocol, without the actual reactions and protocols assigned to its constituent channels and families, so we have implemented a method that provides a representation of this shape, in our new JSON-like notation.´

4 Omitting Types and Variable Names in Bounds

In general binds are of the form `x : T <- R`, and a bind typechecks if the type of `R` is `T`. We want to be able to omit explicitly giving the type, with the intended meaning of it being the type of `R`, and then the bind typechecks implicitly. Moreover, when we read from a channel but we never use the result, we may choose not to provide a name for the variable used in that bind. In the case studies we have represented this as `_ : T <- R`.

For omitting types, we introduce a new notational variant `x <- R` together with a reaction equality rule that adds the missing type by computing the type of `R` in the current configuration.

For omitting variable names, we introduce the notation `~<- R`, because, as mentioned above, the underline carries special meaning in Maude. We also define a method that takes a reaction and generates variable names for the underscores occuring in it. We make the convention that the user may not write variables starting with `$`. The names that will be generated are of form `$Ncn` where `N` is a natural number and `cn` is the name of the channel where the reaction occurs. By doing this, we no longer have to deal with ordering issues in the normal forms, as the variable names will be distinct.

Remark 4 *We have strategies for adding the missing type for a certain channel, provided as argument. In order to do this for all channels in a protocol, we must generate a strategy that calls it for each of them. This will happen at the level of the transformation from CS to AS, when the protocol is known.*

Remark 5 *We will generate variable names for underscores when going from the CS to AS, when we compute normal forms for reactions. So far we have computed them by hand.*

5 Slight Modifications in Protocols

When writing sym proofs, we must provide the protocol we rewrite from, as it is not unique. Most often it is a variation of the current protocol, and it is very inconvenient to have to provide it again as a whole. In our case studies we encountered three categories of modifications: changing an existing channel/family occuring in the protocol, adding a new internal channel/family and restructuring the protocol. Usually we have a combination of them, for example adding a new internal channel and altering an existing channel to read from it.

We have decided to provide syntactic support for the first two categories. The added value of having a syntax for the third is little, as the user must still write down the entire new protocol structure. Ideally in the future we could have a graphical way of providing this restructuring.

Remark 6 *Since Maude only provides access to the current protocol after a number of rewrites have been applied in a strategy, we will have to apply the SYM rule in such strategies. At this moment, this implies writing a strategy by hand whenever we make a SYM proof. When the CS will be available, the strategy will be generated at the level of the AS.*