Privacy Calculus in Maude

Georgios V. Pitsiladis[1,2]
gpitsiladis@mail.ntua.gr

Petros Stefaneas[2]
petros@math.ntua.gr

[1] National and Kapodistrian University of Athens, Greece

[2] National Technical University of Athens, Greece

Workshop "Rules: Logic and Applications"
National Technical University of Athens, Greece
December 19, 2018

Contents

A brief history of the framework

D. Kouzapas, A. Philippou, and E. Kokkinofta have defined[1][2][3] a framework

1. to statically check
2. processes of Privacy Calculus
3. for compliance to privacy policies.

[1] Dimitrios Kouzapas and Anna Philippou. "Type Checking Privacy Policies in the π-calculus". In: *Formal Techniques for Distributed Objects, Components, and Systems*. Lecture Notes in Computer Science. Springer, Cham, June 2015, pp. 181–195. DOI: 10.1007/978-3-319-19195-9_12.

[2] Eleni Kokkinofta and Anna Philippou. "Type Checking Purpose-Based Privacy Policies in the π-calculus". In: *Web Services, Formal Methods, and Behavioral Types*. Lecture Notes in Computer Science. Springer, Cham, Sept. 2014, pp. 122–142. DOI: 10.1007/978-3-319-33612-1_8.

[3] Dimitrios Kouzapas and Anna Philippou. "Privacy by Typing in the π-calculus". In: *Logical Methods in Computer Science* 13.4 (Dec. 2017). DOI: 10.23638/LMCS-13(4:27)2017.

A brief history of the framework

D. Kouzapas, A. Philippou, and E. Kokkinofta have defined a framework

1. to statically check
2. processes of Privacy Calculus
3. for compliance to privacy policies.

Privacy Calculus is a variant of typed π-calculus, with types expressing privacy-related information.
Programmes of Privacy Calculus are organised in two levels:

1. Processes (describing actions),
2. Systems (adding useful meta-information to Processes: Groups and Purposes).

A brief history of the framework

D. Kouzapas, A. Philippou, and E. Kokkinofta have defined a framework

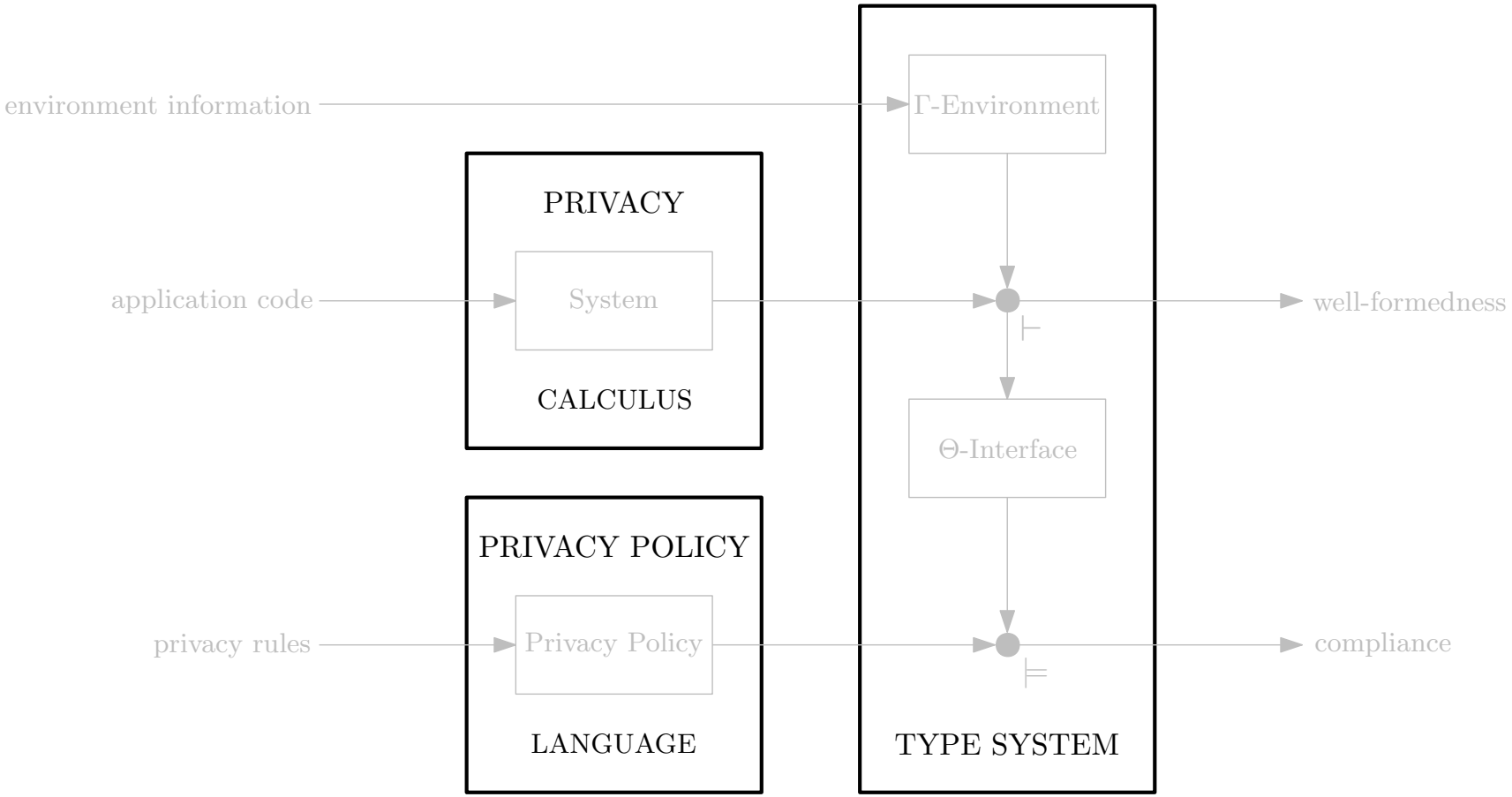
1. to statically check
2. processes of Privacy Calculus
3. for compliance to privacy policies.

We have implemented a version of the framework –extended with conditional permissions[1][2]– in Maude.

[1] Georgios V. Pitsiladis. "Type Checking Privacy Policies in the π-calculus and its Executable Implementation in Maude". Greek. Diploma thesis (supervised by Petros Stefaneas). Greece: National Technical University of Athens, 2016. URL: http://dspace.lib.ntua.gr/handle/123456789/44439.

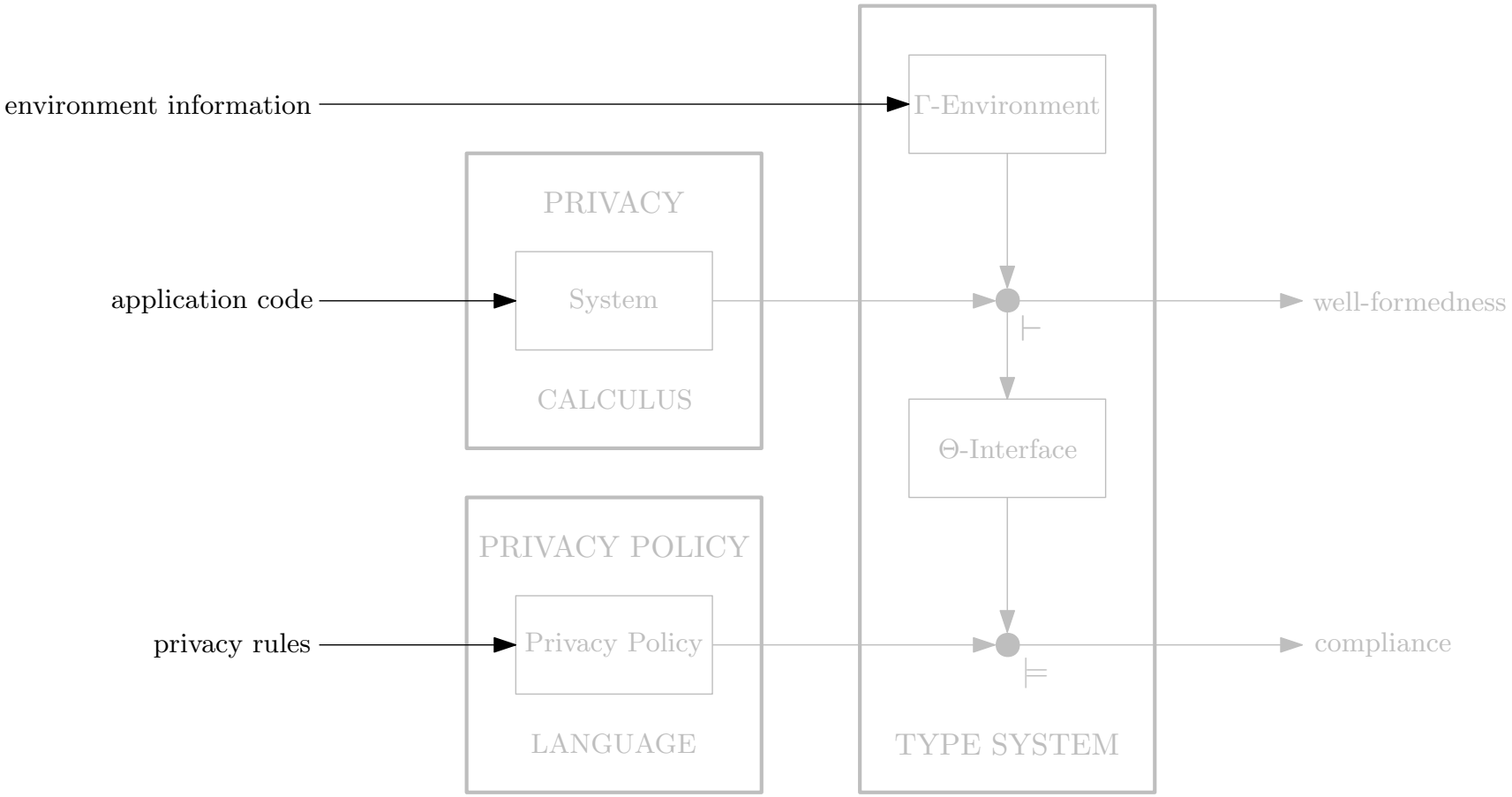
[2] Georgios V. Pitsiladis. "Type Checking Conditional Purpose-Based Privacy Policies in the π-calculus". In: *1st Workshop for Formal Methods on Privacy*. Limassol, Cyprus, 2016. URL: http://users.ntua.gr/gpitsiladis/files/documents/2016-11-fmpriv-conditions.pdf.

The structure of the framework



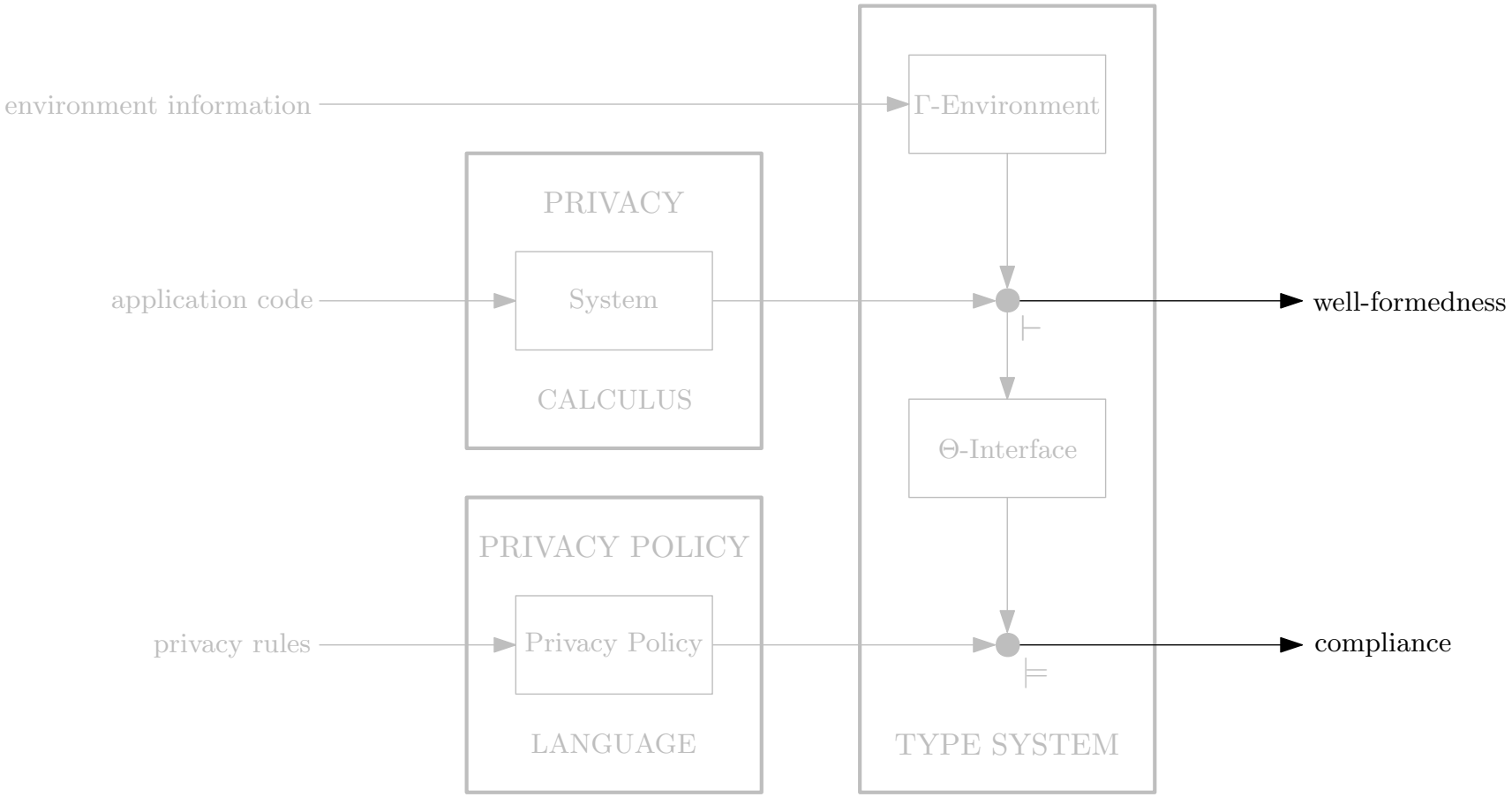
Comprised of three parts: (a) the Privacy Calculus, (b) the Privacy Policy Language, and (c) the Type System.

The structure of the framework



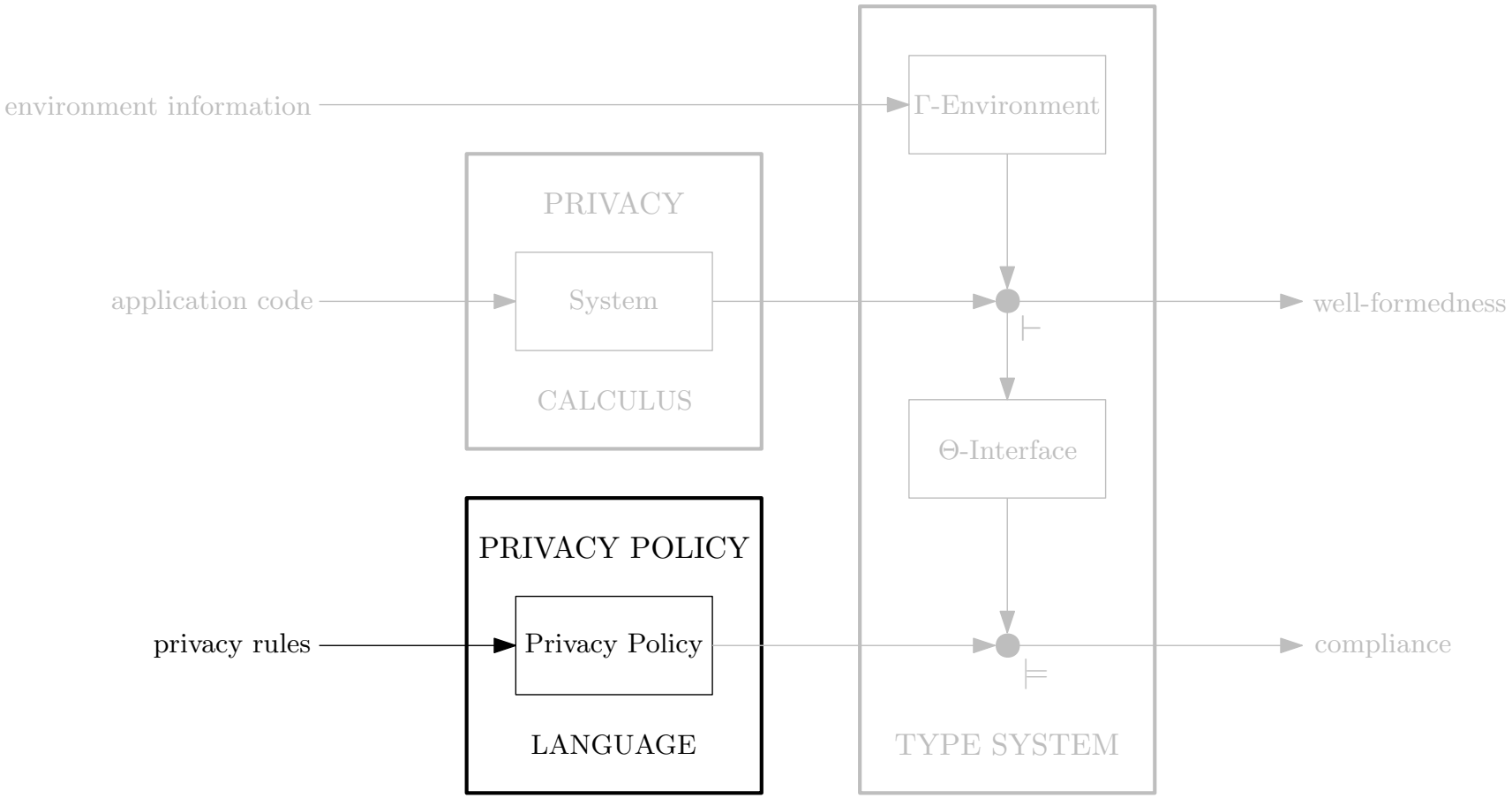
Must be given (a) some privacy rules, (b) the code of an application, and (c) some information regarding the environment of the application.

The structure of the framework



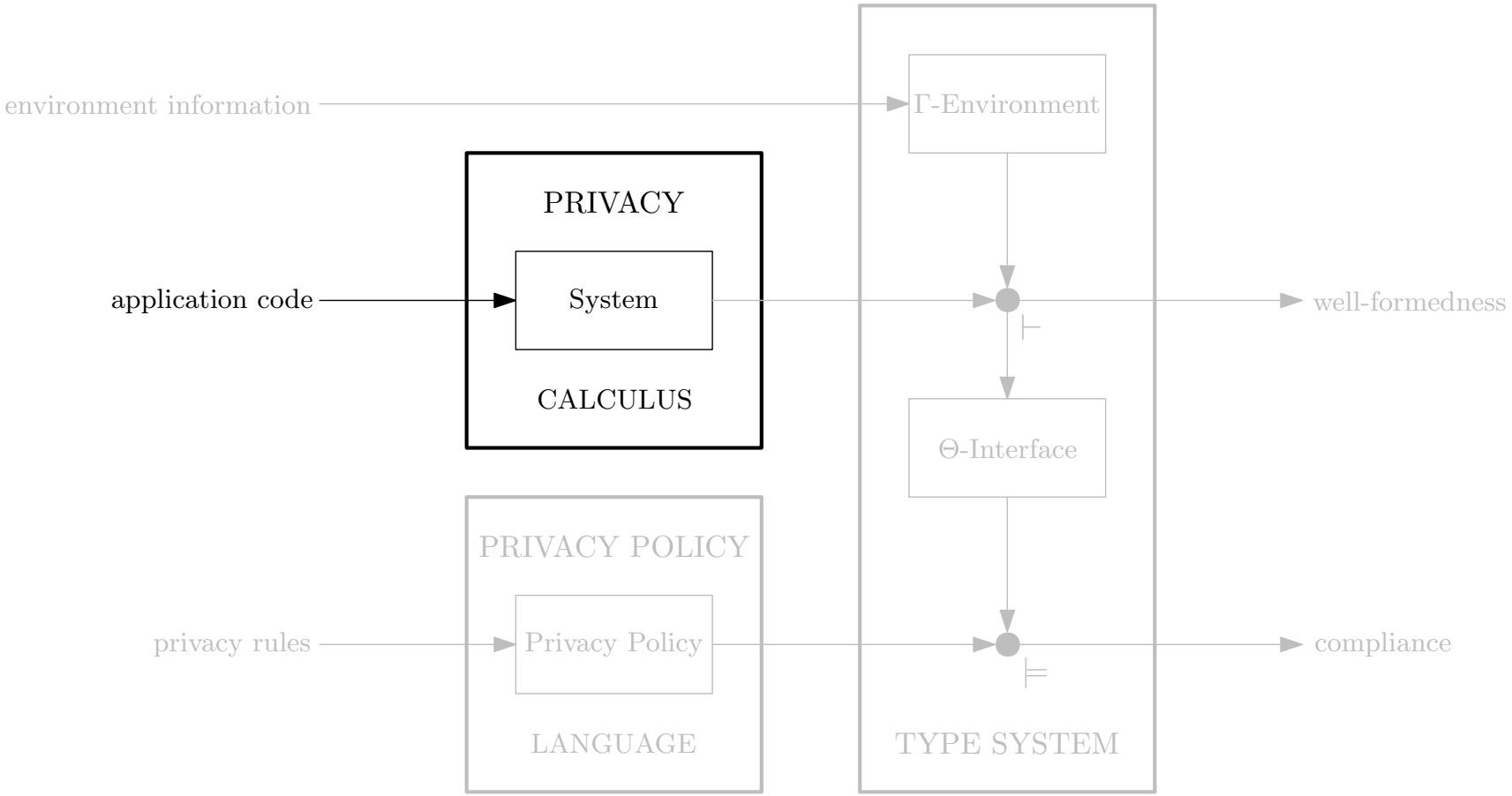
Checks the well-formedness of the given application and its compliance to the given privacy policy.

The structure of the framework



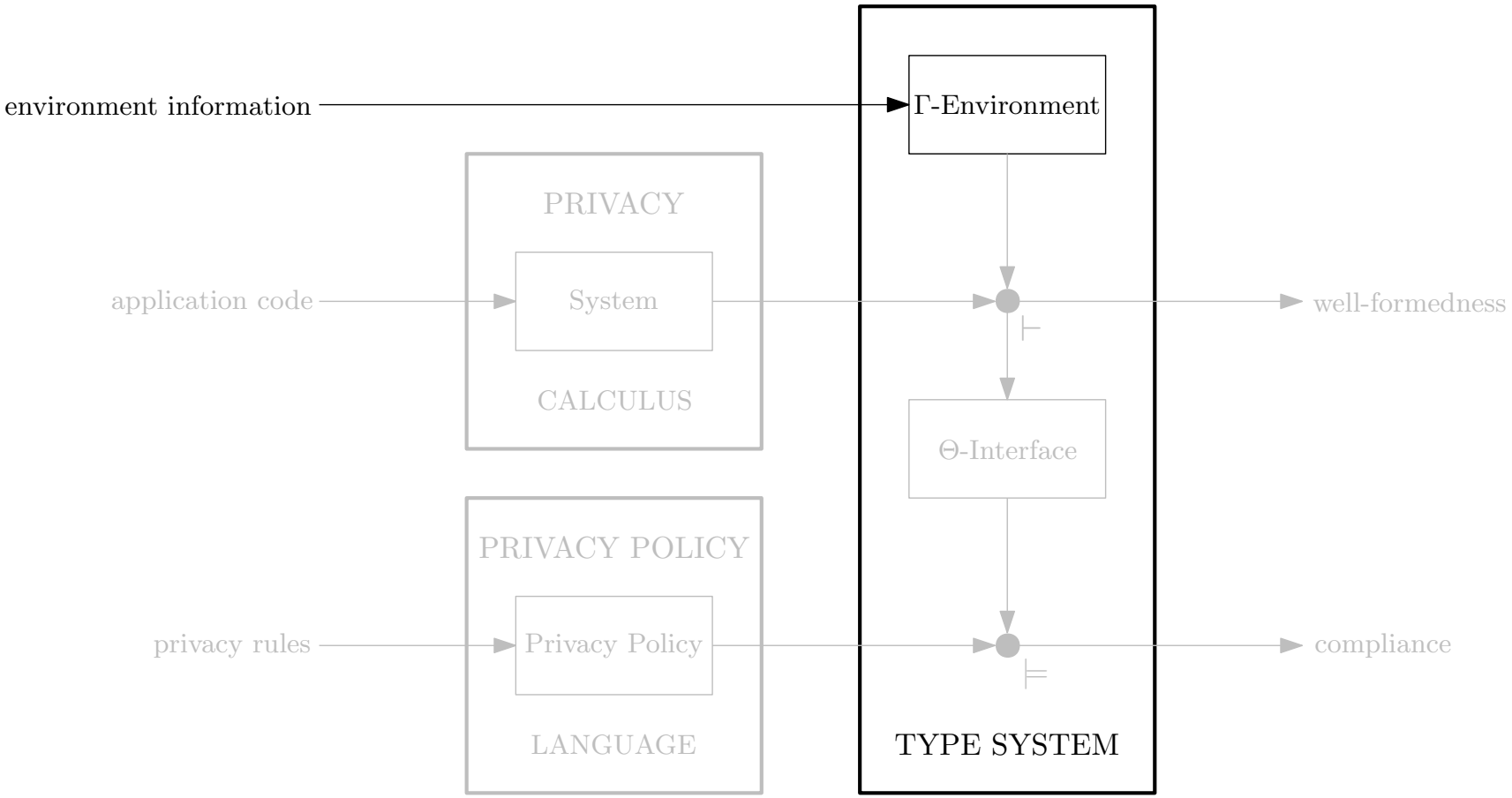
At this point, the privacy rules must be given directly as a Privacy Policy in the formal language of the framework. In the future, they could be "compiled" from higher level information.

The structure of the framework



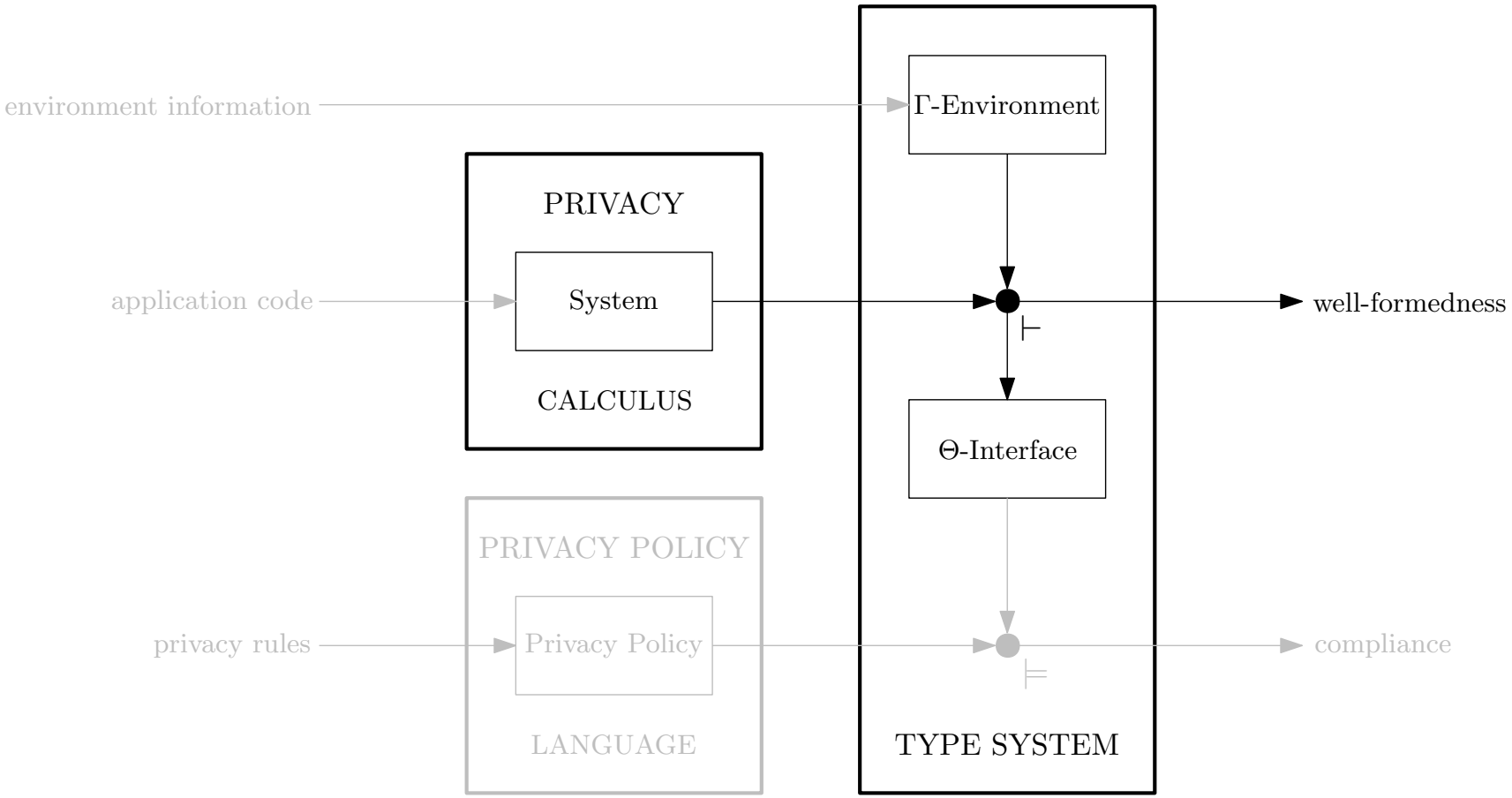
At this point, the application code must be given directly as a System of the Privacy Calculus. In the future, it could be generated from actual application code.

The structure of the framework



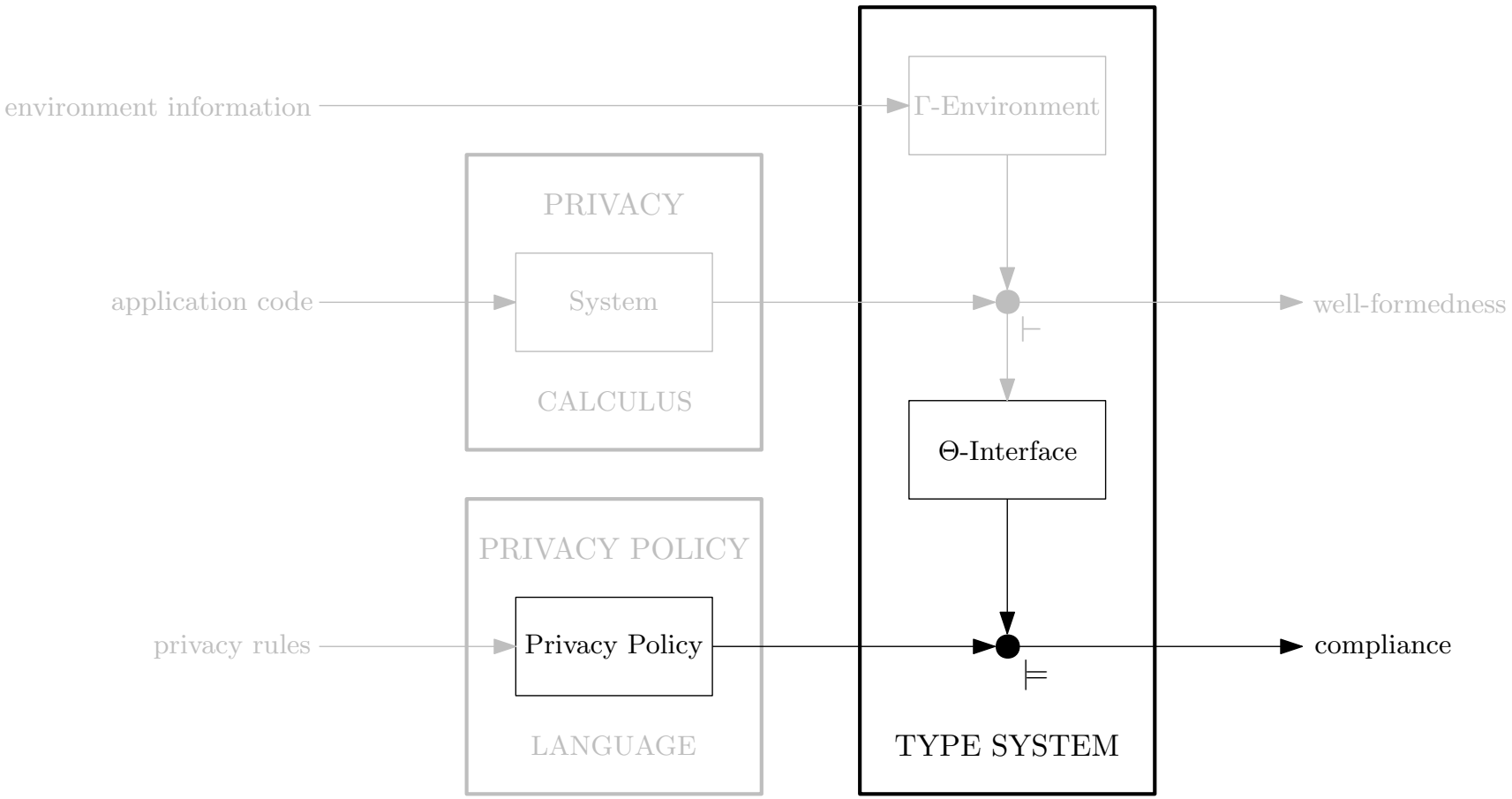
Γ-Environments store information related to the environment. At this point, they store (a) information on public channels of the π-calculus application, (b) known users and groups, and (c) active conditions (eg time, age of user).

The structure of the framework



The type checker (operator $\vdash$), using Γ-Environments (as is standard in type checking), ensures the well-formedness of the Privacy Calculus System and, if it is well-formed, extracts its type (called a Θ-Interface).

The structure of the framework



The extracted Θ-Interface is checked for compliance to the Privacy Policy, using the operator $\models$. Essentially, a Θ-Interface contains structured information on how the System handles private data and $\models$ checks whether the parts of the Θ-Interface can be "mapped" to sub-policies of the Policy.

About Maude

Maude[1] is a specification/programming language with a lot of advantages.

- Solid mathematical foundations: equational and rewriting logic.
- Ability to execute specifications.
- Reflective character: specifications can be handled as data;
  hence, there is the ability to prove properties of specifications written in Maude within Maude itself.
- Efficiency.

[1] Manuel Clavel et al. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. Programming and Software Engineering. Berlin Heidelberg: Springer-Verlag, 2007. ISBN: 978-3-540-71940-3.

Rules in Maude

In Maude, the code is organised into functional and system modules.

Rules in Maude

In Maude, the code is organised into functional and system modules.
Functional modules contain equational rules.
Equational rules substitute equals for equals in order to reduce a term to its (irreducible) canonical form.

eq `fn(NES1 || NES2) = union(fn(NES1), fn(NES2))` .
––– i.e. the free names of parallel stuff is the union of the free names

Rules in Maude

In Maude, the code is organised into functional and system modules.
Functional modules contain equational rules.
Equational rules substitute equals for equals in order to reduce a term to its (irreducible) canonical form.

eq `fn(NES1 || NES2) = union(fn(NES1), fn(NES2))` .
——— *i.e. the free names of parallel stuff is the union of the free names*

System modules contain (equational and) rewriting rules.
Rewriting rules describe (possibly non-deterministic) transitions between (unequal) terms of the same sort.

crl `[Comm] : NES1 || NES2 => {silent} ([A := Z] S1) || S2`
if `NES1 => {in X(A)} S1`
$\wedge$ `NES2 => {out X(Z)} S2` .
——— *i.e.*

$$\frac{NES_1 \xrightarrow{x(a)} S_1 \qquad NES_2 \xrightarrow{\overline{x}\langle z\rangle} S_2}{NES_1 \parallel NES_2 \xrightarrow{\tau} ([a := z]\, S_1) \mid S_2}\ (\text{Comm})$$

Our implementation of the framework in Maude

Our Maude modules[1] implement all the parts of the framework discussed above (as functional modules, i.e. using equational logic).
They are mostly translations of the mathematical definitions.

[1] Available at `http://users.ntua.gr/gpitsiladis/isola2018/privacy.maude`

Our implementation of the framework in Maude

Our Maude modules[1] implement all the parts of the framework discussed above (as functional modules, i.e. using equational logic).
They are mostly translations of the mathematical definitions.

Moreover, our modules also contain the labelled transition semantics of Privacy Calculus (as a system module, i.e. using rewriting logic), so it should be possible to inspect the possible behaviours of a Privacy Calculus System, using the **search** command of Maude.

[1] Available at `http://users.ntua.gr/gpitsiladis/isola2018/privacy.maude`

Some technical considerations

Type checking is implemented as an equational module, due to its deterministic nature.

In the specification of Privacy Calculus Systems, it is necessary to employ the keyword **frozen** of Maude, which restricts rewrites to outmost terms (the roots of syntactic trees) that build Systems.

We use CINNI[1] to handle name binding in Privacy Calculus.

[1] Mark-Oliver Stehr. "CINNI - A Generic Calculus of Explicit Substitutions and its Application to λ- ς- and π-calculi". In: *Electronic Notes in Theoretical Computer Science*. The 3rd International Workshop on Rewriting Logic and its Applications 36 (Jan. 2000), pp. 70–92. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80125-2.

Some technical considerations II

Since Maude rewrites do not support variables in labels, labelled transition semantics needs some workaround[1]: labels must be included in the resulting term of the transition.

```
crl [Comm] : NES1 || NES2 => {silent} ([A := Z] S1) || S2
  if NES1 => {in X(A)} S1
  ∧ NES2 => {out X(Z)} S2 .
```

i.e.

$$\frac{NES_1 \xrightarrow{x(a)} S_1 \qquad NES_2 \xrightarrow{\overline{x}\langle z\rangle} S_2}{NES_1 \parallel NES_2 \xrightarrow{\tau} ([a := z]\, S_1) \mid S_2}\ (\text{Comm})$$

[1] Prasanna Thati, Koushik Sen, and Narciso Martí-Oliet. "An Executable Specification of Asynchronous π-calculus Semantics and May Testing in Maude 2.0". In: *Electronic Notes in Theoretical Computer Science*. WRLA 2002, Rewriting Logic and Its Applications 71 (Apr. 2004), pp. 261–281. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)82539-3.

Using the framework in Maude

All the parts of the specification can be executed in Maude (we tested it in Maude 2.7).

Our Maude modules can be found at
`http://users.ntua.gr/gpitsiladis/isola2018/privacy.maude`

A short example of how the framework could be used in an e-shop is available at
`http://users.ntua.gr/gpitsiladis/isola2018/example-sales.maude`

Future steps using Maude

We have only used a small subset of Maude's power: Core Maude.

Maude's reflection gives a lot of opportunities:

- Use theorem proving in Maude to prove facts about Privacy Calculus and its type checking.
- Use Maude tools[1] such as the Church-Rosser Checker, the Sufficient Completeness Checker, and the Maude Termination Tool to prove that our specification is semantically sound.

[1] Manuel Clavel et al. *Maude Manual (Version 2.7)*. Tech. rep. SRI International Computer Science Laboratory, 2015. URL: http://maude.cs.uiuc.edu/maude2-manual, Sect. 1.3.

Thank you!