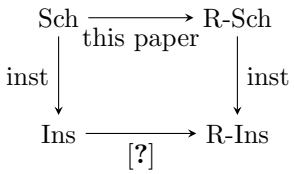

Figure 1: Schema of the proof of correctness.

1 Consistency

In Fig. 3 we can observe the relation occurring between schemas and instances. On the top-left corner we have non-reversible schemas, on the top-right we have reversible schemas, while below on the bottom-right corner we have concrete non-reversible instances - i.e., ground rules - and on the bottom-left corner we have concrete reversible instances.

Schemas differ from instances as they admit the presence of meta-variables that can range over ground terms. Intuitively, to get from one of the top corners to the respective concrete one we need to instantiate each meta-variable with the appropriate ground instance. The correctness of the rule is then ensured by the set of equations each rule is equipped with, if the concrete instances of the meta-variables satisfy the equational condition then the instance of the rule is to be considered correct.

Now, let us discuss the top-arrow, that is transforming a non-reversible schema in reversible schema. A non-reversible schema has the following shape:

$$t \to t' \text{ if } \overline{eq}_n$$

where t and t' are inductively defined as

$$\begin{array}{l} t, t', t_1, \ldots, t_n ::= \\ \quad \mid e \\ \quad \mid op(t_1, \ldots, t_n) \end{array}$$

Now, for a schema to be reversible each entity must be tagged with a unique meta-key and a memory has to be produced each time that a transition is performed. In symbols

$$t_r \to t'_r \mid \mu \text{ if } \overline{eq}_n \text{ where } ukeys(t_r \to t'_r)$$

Where t_r and t'_r are inductively defined as

$$\begin{array}{l} t_r, t'_r, t_{1r}, \ldots, t_{nr} ::= \\ \quad \mid e * k \\ \quad \mid op(t_{1r}, \ldots, t_{nr}) \end{array}$$

and μ is defined as $[R; C]$ where R is the configuration that gave rise to the step, i.e., t_r, and C is a context describing the structure of the resulting one, where entities have been removed and replaced with a hole $\bullet$, while keys are kept.

In symbols C is inductively defined as

$$\begin{array}{l} c, c', c_1, \ldots, c_n ::= \\ \quad \mid \bullet * k \\ \quad \mid op(c_1, \ldots, c_n) \end{array}$$

Let us define the *tag* operation

$$\begin{array}{l} tag(t \to t') ::= \\ \quad \mathsf{let}(t_r, keys) = tag\_left(t, K)\ \mathsf{in} \\ \quad \mathsf{let}(t'_r, \_, \_) = tag\_right(t', keys, 0)\ \mathsf{in} \\ \quad\quad t_r \to t'_r \\ tag\_left(t, k :: keys) ::= \\ \quad \mathsf{match}\ t\ \mathsf{with} \\ \quad\quad \mid e \to (e * k', k' :: k :: keys) \\ \quad\quad \mid op(tlist) \to \\ \quad\quad\quad \mathsf{let}\ f = \mathsf{fun}(x, (rtlist, keys')) \to \mathsf{let}(x_r, keys'') = tag\_left(x, keys')\ \mathsf{in}\ (rtlist :: x_r, keys'')\ \mathsf{end\ in} \\ \quad\quad\quad \mathsf{let}\ (rtlist, keys') = \mathsf{foldl}\ tlist\ ([], keys)\ f\ \mathsf{in} \\ \quad\quad\quad\quad (op(rtlist), keys') \\ tag\_right(t, keys, c) ::= \\ \quad \mathsf{match}\ t\ \mathsf{with} \\ \quad\quad \mid e \to \mathsf{let}\ c' = c + 1\ \mathsf{in} \\ \quad\quad\quad (e * c' :: keys, c') \\ \quad\quad \mid op(tlist) \to \\ \quad\quad\quad \mathsf{let}\ f = (x, (rtlist, C, keys)) \to \mathsf{let}(x_r, C') = tag\_right(x, keys, C)\ \mathsf{in}\ (rtlist :: x_r, C', keys)\ \mathsf{end\ in} \\ \quad\quad\quad \mathsf{let}\ (rtlist, C, keys') = \mathsf{foldl}\ tlist\ ([], c, keys)\ f\ \mathsf{in} \\ \quad\quad\quad\quad (op(rtlist), C, keys') \end{array}$$

The function *tag* takes in input a non reversible schema and produces a tagged version of the schema, where each entity is tagged with a unique meta-key. The meta-keys used to tag the entities must be unique since then in the concrete instances each entity has to be tagged with a unique concrete key, if in the schema two (or more) entities would be tagged with the same meta-key then it would be impossible to instantiate them to different values.

In order to simplify the proof of the correctness of the transformation instead of the function *tag* we will use relation $\rightsquigarrow_k$ to abstract the transformation of terms. Relation $\rightsquigarrow_k$ is the least relation that satisfies the following rules.

$$\frac{\mathsf{lw}(e) \quad k \text{ is fresh}}{e \rightsquigarrow_k e * k} \qquad \frac{t_1 \rightsquigarrow_k t'_1 \ \ldots \ t_n \rightsquigarrow_k t'_n}{op[t_1, \ldots, t_n] \rightsquigarrow_k op[t'_1, \ldots, t'_n]}$$

Relation $\rightsquigarrow_k$ faithfully describes the behavior of both *tag_right* and *tag_left* as the rule on the left corresponds to the base case, where a fresh key is attached to an entity of the lower level and the rule on the right corresponds to the

recursive call on the terms used by operator op. The predicate lw makes sure that e belongs to the lower level of the semantics.

Once a non-reversible schema has been tagged we need to produce the memory, for that we rely on the function mem, that given a rule returns the rule together with the appropriate memory.

$$
\begin{array}{l}
mem(t_r \to t'_r) ::= \\
\quad \mathsf{let}\ C = ctx(t'_r)\ \mathsf{in} \\
\quad t_r \to t'_r \mid [C; t_r] \\
\\
ctx(t) ::= \\
\quad \mathsf{match}\ t\ \mathsf{with} \\
\quad\quad |e * k \to \bullet * k \\
\quad\quad |op(\overline{t_r}) \to \\
\quad\quad\quad \mathsf{let}\ f = \mathsf{fun}(x, \overline{c_m}) \to\ \overline{c_m} :: ctx(x)\ \mathsf{end}\ \mathsf{in} \\
\quad\quad\quad \mathsf{let}\ \overline{c_n} = \mathsf{foldl}\ (\overline{t_r})\ [\,]\ f\ \mathsf{in} \\
\quad\quad\quad\quad op(\overline{c_n})
\end{array}
$$

To abstract function ctx we rely on the least relation that satisfies $\rightsquigarrow_h$.

$$
\frac{}{e * k \rightsquigarrow_h \bullet * k} \qquad \frac{t_1 \rightsquigarrow_h t'_1\ \ldots\ t_n \rightsquigarrow_h t'_n}{op[t_1, \ldots, t_n] \rightsquigarrow_h op[t'_1, \ldots, t'_n]}
$$

Now we can define the transformation of a non-reversible rule $t \to t'$ to (one of) its reversible counterparts in terms of $\rightsquigarrow$.

$$
\frac{t \rightsquigarrow_k t_r \quad t' \rightsquigarrow_k t'_r \quad t'_r \rightsquigarrow_h ctx}{t \to t'\ \mathsf{if}\ C \rightsquigarrow t_r \to t'_r | [ctx; t_r]\ \mathsf{if}\ C}
$$

$$
\frac{t \rightsquigarrow_k t_r \quad t' \rightsquigarrow_k t'_r \quad t'_r \rightsquigarrow_h ctx}{t \to t'\ \mathsf{if}\ C \rightsquigarrow t'_r | [ctx; t_r]\ \leftarrow t_r}
$$

Now, for the transformation to be correct two ingredients are needed: i) that each entity is tagged with a unique meta-key and ii) that the context produced is correct in the sense that each entity of the right-hand side reversible term has been replaced by a hole.

Lemma 1 (Unicity of keys)**.** Relation $\rightsquigarrow_k$ tags each entity of a term t with a unique key.

Proof. We proceed by induction on the structure of t.
Case $t = e$.
The case of single entity is easy as the entity is left untouched and tagged with a fresh key.
Case $t = op[t_1, \ldots, t_n]$.
By inductive hypothesis we know that entities inside each t'_i have been already tagged with fresh keys, where $t_i \rightsquigarrow_k t'_i$ for $1 \leq i \leq n$, so we can conclude that $op[t'_1, \ldots, t'_n]$ is tagged with unique keys. □

$$\frac{replace(r, v, c_v), \mathcal{I} \rightarrow_I r' \quad}{r, \{(v, c_v)\} \cup \mathcal{I} \rightarrow_I r'} \qquad \frac{var(lhs(r)) = \emptyset}{r, \mathcal{I} \rightarrow_I r}$$

$$\frac{r, \mathcal{I} \rightarrow_I r' \quad E \vdash r' = t \rightarrow t' \text{ if } C}{r \rightsquigarrow_{\mathcal{I},E} t \rightarrow t'}$$

Figure 2: Instantiation relation

Lemma 2 (Correctness of Context)**.** Relation $\rightsquigarrow_h$, given a term t, produces the correct context, that is the same system where all the entities have been replaced by a hole $\bullet$.

Proof. We proceed by induction on the structure of t.
Case $t = e$.
The case of single entity is easy as the entity is replaced with a hole and the key left untouched.
Case $t = op[t_1, \ldots, t_n]$.
By inductive hypothesis we know that each t'_i has been already transformed and entities have been replaced by holes, where $t_i \rightsquigarrow_k t'_i$ for $1 \leq i \leq n$, so we can conclude that $op[t'_1, \ldots, t'_n]$ is the correct context. □

By lemmas 1 and 2 we can conclude that $\rightsquigarrow$ is a correct transformation.

2 From Schemas to Ground Rules

The last part to show is that our schemas can be instantiated to the format expected by the general approach. To achieve so we rely on an instantiation relation that replaces the variables present in the schemas with concrete values.

Before showing the instantiation we recall that in Maude a rule has the generic following shape:

$$t \rightarrow t' \text{ if } C$$

and that the variables used in the rule are not exclusively only the variables introduced in t but that new variables can be introduced in C as well.

We define the instantiation of a schema to a ground rule in terms of the least relation that satisfies the set of rules depicted in Fig. 2.

Relation $\rightarrow_I$ is the relation that replaces variable inside a rule, when all the variables have been replaced the bottom case is reached. Relation $\rightsquigarrow$ relies on relation $\rightarrow_I$ to get a version of rule r where all the vars of the rule have been substituted with concrete values, except for the ones declared in the conditional branch.

The set $\mathcal{I}$ is assumed to be built as follow:

$$\mathcal{I} = \{(v, c_v) | v \in var(lhs(r)), sort(c_v) = sort(v)\}$$

Here, *var* returns the set of variables used inside a term, *lhs* returns the left-hand side term of a rule and *sort* the sort of the variable fed as input. We

also assume that each rule is equipped with a conditional branch, where in the case of unconditional rules - like the backward rules - we assume them to be equipped with a dummy condition, like $true = true$.

Finally, when all the variables, introduced in the left-hand side term, have been replaced everywhere the rule is brought to a canonical form, by relying on the equations of the rewriting logic ($E \vdash r = t \rightarrow t'$ if C).

Let us make this more concrete with an example.

Example 2.1. Let us consider the meta-rule `tau` and the equation about matching.

```
crl [sys-tau] :
    < P | exp: EXSEQ, env-stack: ENV, ASET >  =>
    < P | exp: EXSEQ', env-stack: ENV', ASET >
    if < tau, ENV', EXSEQ' > :=
       < req-gen, ENV, EXSEQ > .

eq [match] :
 < REQLABEL, ENVSTACK, GVALUE = GVALUE > =
 < tau, ENVSTACK, GVALUE > .
```

Then let us consider the following $\mathcal{I}$:

$$\{(\texttt{EXSEQ}, \texttt{true = true}), (\texttt{P}, 2), (\texttt{ENV}, \{\})\}$$

For the sake of simplicity we will ignore the components of a process inside `ASET` as they are not relevant for this example (`ASET` currently only contains the definitions of the functions that the process is allowed to invoke).

Now, after substituting the variables with their current values we obtain a rule with the following shape.

```
crl [sys-tau] :
    < 2 | exp: true = true, env-stack: {}, _ >  =>
    < 2 | exp: EXSEQ', env-stack: ENV', _ >
    if < tau, ENV', EXSEQ' > :=
       < req-gen, {}, true = true > .
```

Then, the only equation that can be applied is equation `match` depicted above and after its application the rule will have the following shape.

```
crl [sys-tau] :
    < 2 | exp: true = true, env-stack: {}, _ >  =>
    < 2 | exp: true, env-stack: {}, _ >
    if < tau, {}, true > :=
       < req-gen, {}, true = true > .
```

Finally, since we have instantiated all the variables with ground values and verified the conditional branch with the equational theory we can drop it and obtain a concrete rule that fits the format required by the general approach.

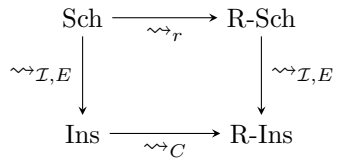

Figure 3: Schema of the proof of correctness.

```
< 2 | exp: true = true, env-stack: {}, _ >  =>
< 2 | exp: true, env-stack: {}, _ >
```

2.1 Commutativity

Before showing that the square commutes we need to define an auxiliary function which serves us to map the keys of a rule to a general format to the format imposed by the schemas. Such function is defined as follow:

$$
\begin{array}{l}
\phi(t \to t'|[ctx;t]) := \\
\quad \mathsf{let}\ K = firstKey(t)\ \mathsf{in} \\
\quad t \to \phi(t,K,0)|[\phi(ctx,K,0);t] \\
\phi(t'|[ctx;t] \to t) := \\
\quad \mathsf{let}\ K = firstKey(t)\ \mathsf{in} \\
\quad \phi(t',K,0)|[\phi(ctx,K,0);t] \to t \\
\phi(t,K,C) := \\
\mathsf{match}\ t\ \mathsf{with} \\
\quad |e * k \to \langle e * key(C :: K), C+1\rangle \\
\quad |\bullet * k \to \langle e * key(C :: K), C+1\rangle \\
\quad |op[tlist] \to \\
\quad\ \mathsf{let}\ \langle tlist', C\rangle = \mathsf{foldl}\ tlist\ ([],C)\ \phi\ \mathsf{in} \\
\quad\ op[tlist']
\end{array}
$$

In words, ϕ is the function that given a reversible rule (either forward or backward) maps the keys of the term running in parallel with the memory from their old values to new values so that the format imposed by the schema is respected.

Lemma 3. The square, up-to keys, commutes, i.e., $\rightsquigarrow_r \rightsquigarrow_{\mathcal{I},E} \overset{\phi}{=} \rightsquigarrow_{\mathcal{I},E} \rightsquigarrow_C$

Proof. We begin by proving that if $(s, gr) \in \rightsquigarrow_r \rightsquigarrow_{\mathcal{I},E}$ then $(s, gr) \in \rightsquigarrow_{\mathcal{I},E} \rightsquigarrow_C$. First, we know that it exist a ground version of s where variables have been replaced with the same ground values of gr, so $(s, g) \in \rightsquigarrow_{\mathcal{I},E}$. Then, since the general approach imposes no constraints on how to pick keys, it must also exist a reversible ground instance of the rule which coincides with gr, in other words, $(g, gr) \in \rightsquigarrow_C$, so we can conclude.

Now, we need to prove that if $(s, gr) \in \rightsquigarrow_{\mathcal{I},E} \rightsquigarrow_C$ then there is a gr_k so that $(s, gr_k) \in \rightsquigarrow_r \rightsquigarrow_{\mathcal{I},E}$, where $\phi(gr) = gr_k$. First, we know that it exists a reversible version of the schema, so we have $(s, s_r) \in \rightsquigarrow_k$. Then, we can proceed to instantiate s_r to gr_k where the ground values are equal to gr except (potentially) for the keys used on the term running in parallel with the memory and hence also on the context. Nonetheless, by applying function ϕ to gr we know that we can always recover the format imposed by the schema, so we can conclude.

□