Bart's syntax and semantics

Lorenzo Bettini[1], Rosario Pugliese[1], and Francesco Tiezzi[1]

Dipartimento di Statistica, Informatica, Applicazioni, Università degli Studi di Firenze, Italy
{lorenzo.bettini,rosario.pugliese,francesco.tiezzi}@unifi.it

Abstract

We provide in this technical report the formal foundations of Bart. We present the syntax in Sec. 1. We first define the semantics in Sec. 2 so that the final result of evaluating a request against a policy system is a boolean value. Then, in Sec. 3, we extend the semantics so that, the evaluation also returns the authorized requests generated during the evaluation.

1 Syntax

The syntax of Bart is reported in Table 1. It is given through an EBNF-like grammar, where, as usual, the symbol $*$ stands for (possibly empty) sequences, $+$ for non-empty sequences, and ? for an optional element.

A top-level term is a Policy System (*PS*) defined by a sequence of policies (*Policy*$^+$). A policy (*Policy*) defines the access rules for the resources controlled by a single party of the policy system. It is defined as a pair of labeled fields: the first field (party) specifies a non-empty list of attributes (*Attribute*$^+$) that characterize the party and permit its identification, while the second field (rules) specifies a list of access rules (*Rule**). Notably, the empty list of rules represents a party that manages no resource but can make requests to access resources managed by other parties.

An attribute (*Attribute*) is a name-value pair, where *name* is a string that refers to the literal *value* associated with the attribute; *name* and *value* are generic elements for the set of names and values, respectively. The syntax of values is deliberately omitted; we assume to use booleans, integers, doubles, strings, dates, and sets of those values. We assume no *name* occurs multiple times as an attribute name in any attribute list.

A rule (*Rule*) is defined as a triple of labeled fields: the first field (resource) specifies a non-empty list of attributes characterizing a resource controlled by the rule's owner party; the second field (condition) specifies an expression (*Expr*) indicating the condition that must be satisfied for granting the access to the resource; the third field (exchange), if present, specifies which accesses to which resources for which parties the rule's owner party requires in exchange for granting access to the resource.

The syntax of expressions (*Expr*) is deliberately left underspecified; we just assume that expressions are built from attribute names and values by using standard logical, relational, and arithmetic operators. Notably, attribute names in expressions can refer to attributes specified in the request under evaluation or the requester's field party or the context (the notion of context will be introduced in Sec. 2).

An exchange (*Exchange*) is a combination using the and and or Boolean operators of triples of labeled fields. The first field (to) of such a triple specifies to which parties (*To*) the resource access in exchange has to be granted; the second field (resource) identifies a resource, as usual, in terms of a list of attributes; the last field (from) specifies which parties (*From*) should grant resource access in exchange. The rule's owner party can specify that the resource access in exchange has to be granted to itself (me) or other parties (*Others*) and can specify as well that the resource access in exchange has to be provided by the party that sent the access request (requester) or other parties (*Others*). The other parties can be identified using a list of attributes by selecting any (anySuchThat) or all (allSuchThat) parties in the policy system that satisfies them. Thus, exchanges can specify generic conditions of access to resources, not necessarily requiring access only for the rule's owner.

Table 1: User syntax of Bart

PS	$::=$	Policy^{+}
Policy	$::=$	$(\mathsf{party} : \mathit{Attribute}^{+}, \mathsf{rules} : \mathit{Rule}^{*})$
$\mathit{Attribute}$	$::=$	$(\mathit{name} : \mathit{value})$
Rule	$::=$	$(\mathsf{resource} : \mathit{Attribute}^{+}, \mathsf{condition} : \mathit{Expr}, \mathsf{exchange} : \mathit{Exchange}^{?})$
Expr	$::=$	$\mathit{name} \mid \mathit{value} \mid \mathit{Expr}\ \mathsf{and}\ \mathit{Expr} \mid \mathit{Expr}\ \mathsf{or}\ \mathit{Expr}$
		$\mid \mathsf{not}\ \mathit{Expr} \mid \mathit{Expr} = \mathit{Expr} \mid \mathit{Expr} > \mathit{Expr} \mid \mathit{Expr} + \mathit{Expr} \mid \ldots$
$\mathit{Exchange}$	$::=$	$(\mathsf{to} : \mathit{To}, \mathsf{resource} : \mathit{Attribute}^{+}, \mathsf{from} : \mathit{From})$
		$\mid \mathit{Exchange}\ \mathsf{and}\ \mathit{Exchange} \mid \mathit{Exchange}\ \mathsf{or}\ \mathit{Exchange}$
To	$::=$	$\mathsf{me} \mid \mathit{Others}$
From	$::=$	$\mathsf{requester} \mid \mathit{Others}$
Others	$::=$	$\mathsf{anySuchThat} : \mathit{Attribute}^{*} \mid \mathsf{allSuchThat} : \mathit{Attribute}^{*}$
$\mathit{Request}$	$::=$	$(\mathsf{resource} : \mathit{Attribute}^{+}, \mathsf{from} : \mathit{Others})$

Table 2: Auxiliary syntax of Bart

$\mathit{EnrichedRequests}$	$::=$	$i : \mathit{Request}$
$\mathit{PointToPointRequests}$	$::=$	$i : (\mathsf{resource} : \mathit{Attribute}^{+}, \mathsf{from} : j)$
$\mathit{Context}$	$::=$	$(\mathit{Attribute}^{*})^{+}$

A user request (*Request*) consists of a pair of labeled fields specifying the sequence of attributes that identify the resource to access (field resource), and the target parties (field from) which should provide the resource access.

2 Formal Semantics

We define the semantics of Bart by following a denotational approach through semantic functions mapping each Bart syntactic construct to an appropriate value. The semantics formalizes the process of evaluating access requests over policy systems. During evaluation, each policy (i.e., party) is univocally identified by the index corresponding to its position in the policy system.

To define the semantics, we exploit the auxiliary syntactic constructs reported in Table 2. Specifically, in addition to the (user) requests that only contain the information explicitly specified by the requester user according to the syntax given in Table 1, we consider two kinds of requests. An *enriched (user) request* is a user request enriched with the requester's party index. We assume that the underlying system that manages the user requests automatically and transparently adds this information. A *point-to-point request* is an enriched request where the field from contains a party index representing one of the parties resulting from evaluating the original field from over the policies under consideration. We will use r to range over enriched requests and point-to-point requests. Finally, a *context* (ranged over by c) is expressed as a list of attribute lists representing the contextual attributes of each party of the policy system. In fact, the Bart evaluation process is intrinsically context-aware, allowing authorization decisions to depend on dynamic, low-level data or environmental factors such as the current time, geopositioning data, and more. We expect that this contextual data will be automatically retrieved as attributes during the evaluation process through a context handler.

We also utilize the following auxiliary functions and notations. Empty lists are denoted by $(\,)$. The function $\mathsf{indexes}(t^{*})$ returns the set of indexes of the argument list of terms t^{*}; that is, $\emptyset$ if $t^{*} = (\,)$, or $\{1,\ldots,k\}$ if $t^{*} = t_1 \ldots t_k$ with $k > 0$. The projection function $t^{+}\!\downarrow_i$ returns the i-th element of its (non-empty) argument list of terms t^+; for example, $PS{\downarrow_1}$ returns the first policy of the policy system PS. The function $\mathsf{party}(Policy)$ returns the non-empty attribute list identifying the party owning $Policy$, that is, the attribute list in the field party.

When convenient, we interpret an attribute list $Attribute^*$ as a function from names to values such that $(Attribute^*)(name) = \mathsf{error}$ when $name$ does not occur as an attribute name in $Attribute^*$. We assume that all expression and exchange operators are strict on error and, at the top level, interpret error as a loss of authorization, i.e., false. We check matching between two attribute lists by the predicate $\mathsf{match}(Attribute^*, Attribute^+)$ that holds *true* when, for any $(name\!:\!value)$ in $Attribute^*$, we have $Attribute^+(name) = value$ or $value \subseteq Attribute^+(name)$. It is worth noticing that $\mathsf{match}((\,), Attribute^+) = true$, for any attribute list $Attribute^+$. The function $\mathscr{M}(Attribute^*, PS)$ returns the set of indexes of those parties in a policy system whose policy attribute list matches a given attribute list, i.e.,
$\mathscr{M}(Attribute^*, PS) = \{i \in \mathsf{indexes}(PS) \mid \mathsf{match}(Attribute^*, \mathsf{party}(PS{\downarrow_i}))\}$.

On requests r, we use the function $\mathsf{requester}(r)$, which returns the index of the party making r; the function $\mathsf{resource}(r)$, which returns the content of the field resource of r; and the function $\mathsf{from}(r)$, which returns either the attribute list or the index of the party specified in the field from of r. We also use the function $\mathsf{attributes}(r, c, PS)$ that returns the content of the field resource of r, the j-th field of c, and the field party of $PS{\downarrow_j}$, with $j = \mathsf{requester}(r)$. The predicate $\mathsf{isAny}(r)$ holds if the field from of r has the form $\mathsf{anySuchThat} : Attribute^*$.

We generalize the binary logical operators $\vee$ and $\wedge$, for which we adopt a *short-circuit* left-first evaluation strategy, to multi-arguments operators or and and. We let the logical disjunction $\mathsf{or}_{i\in I} b_i$ to stand for false if the set of indexes I is $\emptyset$, and $b_k \vee \mathsf{or}_{i\in I'} b_i$ if $I = \{k\} \cup I'$. Similarly, we let the logical conjunction $\mathsf{and}_{i\in I} b_i$ to stand for false if $I = \emptyset$, b_k if $I = \{k\}$, and $b_k \wedge \mathsf{and}_{i\in I'} b_i$ if $I = \{k\} \cup I'$ with $I' \neq \emptyset$. Notably, the set of indexes I used in the above operators could result from the evaluation of a conjunction of conditions on the indexes, e.g., $k \in \{1,4,5\}, k \neq 4$ stands for the set $\{1,5\}$. To simplify the notation, we assume that operators $\mathsf{or}_{i\in I}$ and $\mathsf{and}_{i\in I}$ associate to the right; thus, e.g., we write $\mathsf{or}_{i\in I} A_i \wedge B_i$ instead of $\mathsf{or}_{i\in I}(A_i \wedge B_i)$.

Finally, we use the auxiliary predicate

$$\begin{aligned}\mathsf{comply}(r,R) = \exists r' \in R : \ &(\mathsf{requester}(r) = \mathsf{requester}(r')) \ \wedge \ (\mathsf{from}(r) = \mathsf{from}(r')) \ \wedge \\ &\mathsf{match}(\mathsf{resource}(r), \mathsf{resource}(r'))\end{aligned}$$

that intuitively checks if the request set R contains a request r' complying with a given request r, where the compliance between two requests requires them to have the same requester and target and matching resources.

The main clause defining the semantics of Bart is reported in Table 3. It defines the semantic predicate $[\![[PS]]\!]_{r,c}$, which formalizes the evaluation of the enriched request r over the policy system PS under the context c. It returns *true* to mean that the enriched request is authorized and *false* to mean that it is denied. In $[\![[PS]]\!]_{r,c}$, we assume that PS and c have the same length (i.e., each party has a corresponding, possibly empty, list of contextual attributes) and that the set of names occurring in r, c, and the fields party of PS are disjoint (i.e., there is no conflict among request, context, and party attributes). The predicate $[\![[PS]]\!]_{r,c}$ is defined in terms of the auxiliary predicate $[\![t]\!]^{k,PS}_{r,c,R}$ (defined by the clauses reported in Tables 4, 5 and 6), where the argument t is a syntactic term corresponding to a policy or some of its constituents, r is a point-to-point request, c is a context, R is a set of point-to-point requests, k is the index of the party that owns the policy (or its constituent) under evaluation, and PS is a policy system. In the clauses for $[\![t]\!]^{k,PS}_{r,c,R}$, we omit each of the parameters r, c, R, k, and PS whenever it plays no role.

More specifically, the clause (PS) in Table 3 means that the evaluation of the enriched request r over the policy system PS under the context c amounts to suitably composing the results of evaluating the point-to-point requests r_i, obtained from r by replacing the attribute list in the field from with the index i of a matching party in PS, over the policy $PS{\downarrow_i}$ of party i, provided that i is not (the index of) the party

Table 3: Semantics of Bart: evaluation process - part 1 of 4

$$(\mathrm{PS}) \quad [\![\![PS]\!]\!]_{r,c} = \begin{cases} \mathsf{or}_{i\in F, i\neq \mathsf{requester}(r)} [\![PS\downarrow_i]\!]^{i,PS}_{r_i,c,R} & \textit{if } \mathsf{isAny}(r) \\ \mathsf{and}_{i\in F, i\neq \mathsf{requester}(r)} [\![PS\downarrow_i]\!]^{i,PS}_{r_i,c,R} & \textit{otherwise} \end{cases}$$

$$\textit{where } R=\emptyset,\ F \triangleq \mathscr{M}(\mathsf{from}(r),PS), \forall i \in F : r_i \triangleq \mathsf{requester}(r) : (\mathsf{resource} : \mathsf{resource}(r),\ \mathsf{from} : i)$$

Table 4: Semantics of Bart: evaluation process - part 2 of 4

(Pol)	$[\![(\mathsf{party} : \textit{Attribute}^{+}, \mathsf{rules} : \textit{Rule}^{*})]\!] = \mathsf{or}_{i\in \mathsf{indexes}(\textit{Rule}^{*})} [\![\textit{Rule}^{*}\downarrow_i]\!]$
(Rule$_1$)	$[\![(\mathsf{resource} : \textit{Attribute}^{+},\ \mathsf{condition} : \textit{Expr})]\!]_r = \mathsf{match}(\mathsf{resource}(r), \textit{Attribute}^{+}) \wedge [\![\textit{Expr}]\!]$
(Rule$_2$)	$[\![\![(\mathsf{resource} : \textit{Attribute}^{+},\ \mathsf{condition} : \textit{Expr},\ \mathsf{exchange} : \textit{Exchange})]\!]\!]_{r,R} = \mathsf{match}(\mathsf{resource}(r), \textit{Attribute}^{+}) \wedge [\![\textit{Expr}]\!] \wedge [\![\textit{Exchange}]\!]_{R\cup\{r\}}$
(Exp$_1$)	$[\![\textit{name}]\!]^{PS}_{r,c} = (\mathsf{attributes}(r,c,PS))(\textit{name})$
(Exp$_2$)	$[\![\textit{value}]\!] = \textit{value}$
(Exp$_3$)	$[\![\textit{Expr}_1 \mathsf{\ and\ } \textit{Expr}_2]\!] = [\![\textit{Expr}_1]\!] \wedge [\![\textit{Expr}_2]\!]$
(Exp$_j$)	... *clauses for other expression operators* ...
(Exc$_1$)	$[\![\textit{Exchange}_1 \mathsf{\ and\ } \textit{Exchange}_2]\!] = [\![\textit{Exchange}_1]\!] \wedge [\![\textit{Exchange}_2]\!]$
(Exc$_2$)	$[\![\textit{Exchange}_1 \mathsf{\ or\ } \textit{Exchange}_2]\!] = [\![\textit{Exchange}_1]\!] \vee [\![\textit{Exchange}_2]\!]$

that sent r. These results are combined using logical disjunction (resp., conjunction) if the predicate $\mathsf{isAny}(r)$ holds (resp. does not hold). The set R, which is initially empty, is used during the evaluation to check which point-to-point requests have already been encountered, thus avoiding going into vicious exchange circles.

Let us now comment on the clauses reported in Table 4. Evaluating a point-to-point request over a policy (clause (Pol)) amounts to evaluating it over the policy rules and then combining the results using logical disjunction. This implies that for the policy to authorize the request, it is sufficient that one of its constituent rules authorizes it. The policy's attribute list plays no role here but enables exchange constructs to select the parties matching a given attribute list (see clauses in Tables 5 and 6). The evaluation of a point-to-point request r over a rule (clauses (Rule$_1$) and (Rule$_2$)) returns the logical conjunction of the results of the matching between r's attribute list and the rule's attribute list, the evaluation of r over the rule's condition, and, possibly, the evaluation of r over the rule's exchange. Notice that when evaluating r over the exchange, the current point-to-point request r is added to the set R. The evaluation of a point-to-point request over an expression (clauses (Exp$_h$), with $h \geq 1$) proceeds compositionally until either a value is encountered, which is returned as it is, or an attribute is encountered, in which case the result is the value associated with the attribute in the request or in the context or in the field party of the requester. The evaluation of a point-to-point request over a composed exchange construct (clauses (Exc$_1$) and (Exc$_2$)) is defined homomorphically.

The evaluation of a point-to-point request r over a single exchange construct is defined by the clauses in Tables 5 and 6. The general intuition is that if the exchange construct, based on the content of the field to, does not identify any parties to which access has to be granted in exchange, then it means that there are no exchange conditions, and the evaluation returns *true*. Otherwise, the requests generated by the exchange based on the content of its fields are evaluated over the policies of the granting parties identified through the content of the field from, and the results are combined accordingly. Notably, for

Table 5: Semantics of Bart: evaluation process - part 3 of 4

$$(\mathrm{Exc}_3)\quad [\![(\mathsf{to}:\ \mathsf{me},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ \mathsf{requester})]\!]^{k,PS}_{r,R} = \mathsf{comply}(r_{k,i},R) \vee [\![PS\!\downarrow_i]\!]^{i}_{r_{k,i}}$$

$$(\mathrm{Exc}_4)\quad [\![(\mathsf{to}:\ \mathsf{anySuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ \mathsf{requester})]\!]^{PS}_{r,R} = (T=\emptyset) \vee \mathsf{or}_{k\in T, k\neq i}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}}$$

$$(\mathrm{Exc}_5)\quad [\![(\mathsf{to}:\ \mathsf{allSuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ \mathsf{requester})]\!]^{PS}_{r,R} = (T=\emptyset) \vee \mathsf{and}_{k\in T, k\neq i}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}}$$

where $i = \mathsf{requester}(r)$, $T \triangleq \mathscr{M}(\mathit{Attribute}^{*}_{t}, PS)$, *and* :

- *in clause* (Exc_3), $r_{k,i} \triangleq k : (\mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ i)$
- *in the other clauses*, $r_{k,i} \triangleq k : (\mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ i)$, $\forall k \in T$

any exchange construct, the content of the field resource of the requests generated by the exchange is the same as that of the homonym field of the exchange.

The clauses in Table 5 refer to requests to grant access in exchange that are addressed exclusively to the party i that sent the original request r. Clause (Exc_3) concerns exchanges requesting i to grant access to the party k that owns the policy under evaluation. A point-to-point request $r_{k,i}$ is generated from the exchange by using k as a requester and i as the target party that has to grant access in exchange. The evaluation of $r_{k,i}$ returns *true* if it complies with an already encountered point-to-point request or it is authorized by the target party policy $PS\!\downarrow_i$, and *false* otherwise. The rationale of returning *true* when $\mathsf{comply}(r_{k,i},R)$ holds is that, since R contains a request r' complying with $r_{k,i}$, then (i) r' has been added to R necessarily by application of clause (Rule_2) (in fact, this is the only clause that increases R), (ii) r' is under evaluation: on the basis of the resource required, a policy rule x has been identified that applies, but it remains to check the requests generated by the exchange of the rule, (iii) $r_{k,i}$ is no more demanding than r' as concerns the required resource (see definition of predicate comply), thus the same policy rule would apply and the same requests would be generated, (iv) hence, there is no need to evaluate $[\![PS\!\downarrow_i]\!]^{i}_{r_{k,i}}$, and this prevents going in a vicious circle. The remaining two clauses deal with exchanges requiring to grant access to parties in the set T of those parties matching the attribute list $\mathit{Attribute}^{*}_{t}$ contained in the field to of the exchange construct. They only differ in that their final result is obtained by combining the results of evaluating all the point-to-point requests $r_{k,i}$ (with $k \in T$ and $k \neq i$) via a logical disjunction (Exc_4) or a logical conjunction (Exc_5).

The clauses in Table 6 refer to requests for a set F of parties that must grant resource access in exchange, where F is the set of those parties matching the attribute list $\mathit{Attribute}^{*}_{f}$ contained in the field from of the exchange construct. Clause (Exc_6) (resp. (Exc_7)) concerns exchanges that request some party (resp. all parties) in F other than the party k that owns the policy under evaluation to grant access to k. For each $i \in F$ other than k, a point-to-point request $r_{k,i}$ is generated from the exchange by using k as a requester and i as the target party that must grant access in exchange. The final result is obtained by combining the results of evaluating all the above point-to-point requests via a logical disjunction (resp. conjunction). The remaining clauses deal with exchanges requiring to grant access to parties in the set T of those parties matching the attribute list $\mathit{Attribute}^{*}_{t}$ contained in the field to of the exchange construct. They only differ in that their final result is obtained by combining the results of evaluating all the point-to-point requests $r_{k,i}$, for each pair of different parties $k \in T$ and $i \in F$, in all the four possible ways using logical disjunction and conjunction on the sets of indexes T and F.

Table 6: Semantics of Bart: evaluation process - part 4 of 4

$$
\begin{array}{ll}
(\mathrm{Exc}_6) & [\![(\mathsf{to:me},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from:anySuchThat}:\mathit{Attribute}^{*}_{f})]\!]^{k,PS}_{r,R} \\
& \quad = \mathsf{or}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} \\
(\mathrm{Exc}_7) & [\![(\mathsf{to:me},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from:\ allSuchThat}:\mathit{Attribute}^{*}_{f})]\!]^{k,PS}_{r,R} \\
& \quad = \mathsf{and}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} \\
(\mathrm{Exc}_8) & [\![(\mathsf{to:anySuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+}, \\
& \mathsf{from:anySuchThat}:\mathit{Attribute}^{*}_{f}\ )]\!]^{PS}_{r,R} \\
& \quad = (T=\emptyset) \vee \mathsf{or}_{k\in T}\mathsf{or}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} \\
(\mathrm{Exc}_9) & [\![(\mathsf{to:anySuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+}, \\
& \mathsf{from:allSuchThat}:\mathit{Attribute}^{*}_{f}\ )]\!]^{PS}_{r,R} \\
& \quad = (T=\emptyset) \vee \mathsf{or}_{k\in T}\mathsf{and}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} \\
(\mathrm{Exc}_{10}) & [\![(\mathsf{to:allSuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+}, \\
& \mathsf{from:anySuchThat}:\mathit{Attribute}^{*}_{f}\ )]\!]^{PS}_{r,R} \\
& \quad = (T=\emptyset) \vee \mathsf{and}_{k\in T}\mathsf{or}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} \\
(\mathrm{Exc}_{11}) & [\![(\mathsf{to:allSuchThat}:\mathit{Attribute}^{*}_{t},\ \mathsf{resource}:\mathit{Attribute}^{+}, \\
& \mathsf{from:\ allSuchThat}:\mathit{Attribute}^{*}_{f}\ )]\!]^{PS}_{r,R} \\
& \quad = (T=\emptyset) \vee \mathsf{and}_{k\in T}\mathsf{and}_{i\in F, i\neq k}\mathsf{comply}(r_{k,i},R) \vee [\![PS\downarrow_i]\!]^{i}_{r_{k,i}}
\end{array}
$$

where $T \triangleq \mathscr{M}(\mathit{Attribute}^{*}_{t}, PS)$, $F \triangleq \mathscr{M}(\mathit{Attribute}^{*}_{f}, PS)$, *and* :

- *in clauses* (Exc_6) *and* $(\mathrm{Exc}_7), r_{k,i} \triangleq k : (\mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ i), \forall i \in F$
- *in the other clauses*, $r_{k,i} \triangleq k : (\mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ i),\ \forall k \in T, \forall i \in F$

3 From boolean evaluation result to set of permitted requests

We present how functions $[\![[\ ]\!]]$ and $[\![\ ]\!]$ extend to return as a result a configuration of the form $\langle b, R\rangle$, where b is a boolean value and R is a set of point-to-point requests. We follow a bottom-up presentation approach and focus on the salient points.

The key point of the extended semantics is the evaluation of exchange. If the exchange is positively evaluated, the generated subordinate requests that have been evaluated positively must be returned. The clause (Exc_3) is thus extended as follows:

$$
[\![(\mathsf{to:me},\ \mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from:requester})]\!]^{k,PS}_{r,R}
= \begin{cases}
\langle \mathit{true}, \emptyset\rangle & \mathit{if}\ \mathsf{comply}(r_{k,i},R) \\
\langle \mathit{true}, R\rangle & \mathit{if}\ \neg\,\mathsf{comply}(r_{k,i},R) \wedge [\![PS\downarrow_i]\!]^{i}_{r_{k,i}} = \langle \mathit{true}, R\rangle \\
\langle \mathit{false}, \emptyset\rangle & \mathit{otherwise}
\end{cases}
$$

where $i = \mathsf{requester}(r)$, and $r_{k,i} \triangleq k : (\mathsf{resource}:\mathit{Attribute}^{+},\ \mathsf{from}:\ i)$. Intuitively, if the compliance relation is satisfied, the exchange evaluation stops, and no additional point-to-point request is returned. Instead, if the compliance relation is not satisfied, request $r_{k,i}$ is evaluated over the appropriate policy and, in case of a positive evaluation, the produced set R is returned. Notice that R contains the request $r_{k,i}$ by construction (see the extension of clause (Pol) below).

The semantics of or operator for exchanges (clause (Exc_2)) extends obviously:

$$[\![\mathit{Exchange}_1 \ \mathsf{or}\ \mathit{Exchange}_2]\!] = \begin{cases} \langle \mathit{true}, R_1\rangle \ \mathit{if}\ [\![\mathit{Exchange}_1]\!] = \langle \mathit{true}, R_1 \rangle \\ \langle \mathit{true}, R_2\rangle \ \mathit{if}\ [\![\mathit{Exchange}_1]\!] = \langle \mathit{false}, R_1 \rangle \wedge [\![\mathit{Exchange}_2]\!] = \langle \mathit{true}, R_2 \rangle \\ \langle \mathit{false}, \emptyset \rangle \ \ \mathit{otherwise} \end{cases}$$

The semantics of rules with exchanges (clause (Rule$_2$)) extends as follows:

$$[\![(\mathsf{resource} : \mathit{Attribute}^+, \mathsf{condition} : \mathit{Expr}, \mathsf{exchange} : \mathit{Exchange})]\!]_{r,R}$$
$$= \begin{cases} \langle \mathit{true}, R' \rangle \ \ \mathit{if}\ \mathsf{match}(\mathsf{resource}(r), \mathit{Attribute}^+) \wedge [\![\mathit{Expr}]\!] \wedge [\![\mathit{Exchange}]\!]_{R \cup \{r\}} = \langle \mathit{true}, R' \rangle \\ \langle \mathit{false}, \emptyset \rangle \ \ \mathit{otherwise} \end{cases}$$

Briefly, in the case of a positive evaluation, the request set produced by the exchange evaluation is the result of the rule evaluation. Notably, evaluating expressions does not need to be extended, and the result is still a boolean value.

The semantics of policies (clause (Pol)) is as follows:

$$[\![(\mathsf{party} : \mathit{Attribute}^+, \mathsf{rules} : \mathit{Rule}^*)]\!]_r = \begin{cases} \langle \mathit{true}, R \cup \{r\} \rangle \ \mathit{if}\ \mathsf{or}_{i \in \mathsf{indexes}(\mathit{Rule}^*)} [\![\mathit{Rule}^* \downarrow_i]\!]_r = \langle \mathit{true}, R \rangle \\ \langle \mathit{false}, \emptyset \rangle \qquad\quad \mathit{otherwise} \end{cases}$$

where:

$$\mathsf{or}_{i \in I} \langle b_i, R_i \rangle = \begin{cases} \langle b_k, R_k \rangle \quad \mathit{if}\ \exists k \in I \ :\ k = \mathit{min}(\{ j \in I \mid b_j = \mathit{true} \}) \\ \langle \mathit{false}, \emptyset \rangle \ \ \mathit{otherwise} \end{cases}$$

The evaluation of a policy consists, as usual, of the disjunction of the results produced by evaluating its rules. The disjunction is evaluated, again, using a short-circuit strategy, i.e., as soon as a positive configuration is encountered, the evaluation stops. It is worth noticing that when a policy is evaluated positively, the set of requests produced by the evaluation of its rules and enriched with the request under evaluation is returned.

Finally, the definition of the top-level clause (PS) does not require any change, except for the use of the extended and operator:

$$\mathsf{and}_{i \in I} \langle b_i, R_i \rangle = \begin{cases} \langle \mathit{true}, \bigcup_{i \in I} R_i \rangle \ \ \mathit{if}\ \forall j \in I \ :\ b_j = \mathit{true} \\ \langle \mathit{false}, \emptyset \rangle \qquad\quad \mathit{otherwise} \end{cases}$$