Evaluating FROST Policies Under Incomplete Information

Technical Note

1 Motivation and Background

A PIP or perhaps even the requester may not be able to provide all the information about attribute or entity values pertaining to the evaluation of a policy pol. This inability may reflect a fault, e.g. that a communication line is temporarily down, but it may alternatively be caused by an active security attack.

It practice it is hard to always reliably distinguish between fault and attack events. Therefore, the FROST technology will take a conservative stance and interpret all such events as potentially malicious. One implication of this is that any attempt of the PDP to recover the policy outcome from such incomplete information should not compute an outcome that makes additional assumptions about such missing information.

Informally, each term in the policy language will have a type that designates the set of values that this term can take on. For example, an $entity$ term may have a digital identity type, saying that its value can only be a bit string of a fixed length. Or a term of type int may range over 64-bit integer values.

Semantically, a model ρ for a policy pol is a function that maps each atomic term t occurring in pol to some value $\rho(t)$ of its domain of values. For example $\rho(x)$ may be a concrete 64-bit integer if x has that type. By atomic terms we mean those that are not composed out of other terms, e.g. $x + 5 \cdot y$ is not atomic. The term $vehicle.owner.daugther$ also is not atomic and it could have three locations of incomplete information (all of which would make this value unknown).

Let us say that we extend each value domain (for each type) with a special symbol $\perp$ which denotes that a value is unknown. In particular, model ρ can now also be *partial* in that it may map some atomic terms to $\perp$. The evaluation of terms under a partial model ρ then extends to the evaluation of atomic conditions, e.g. $\rho(x < 2 \cdot y) = \perp$ if $\rho(x) = \perp$. We define this formally:

Definition 1. *Let AT be a set of atomic terms for the FROST language where each $t \in AT$ can attain any value from a domain Dom^T where T is the type of t. Then we define:*

1. *A model ρ is a total function from AT to $\bigcup_T Dom^T$ where $\rho(t)$ is in Dom^T for all $t \in AT$ with type T.*
2. *A partial model ρ is a total function from AT to $\bigcup_T Dom^T_\perp$ where $Dom^T_\perp$ equals $Dom^T \cup \{\perp\}$ and $\rho(t)$ is in $Dom^T_\perp$ for all $t \in AT$ of type T.*
3. *Partial models are endowed with a partial order $\rho \trianglelefteq \rho'$ iff for all $t \in AT$ we have $\rho(t) = \perp$ or $\rho(t) = \rho'(t)$.*

Importantly, partial models allow us to express incomplete information about a (total) model. As for the partial order $\rho \trianglelefteq \rho'$, it means that ρ' refines ρ since it has the same concrete (i.e. non-$\perp$) values that ρ has, and it may also attain concrete values for atomic terms that are mapped to $\perp$ under ρ. Note that the maximal models η under this partial order are those that have no missing information: the (total) models η satisfying $\eta(t) \neq \perp$ for all atomic terms t.

Definition 2. *Let ρ be a partial model. We write $\mathsf{Compl}\,(\rho)$ for the set of all such η that satisfy $\rho \trianglelefteq \eta$ and are maximal elements in that partial order.*

Intuitively, $\mathsf{Compl}\,(\rho)$ are the *completions* of ρ. This set of non-partial models represents all "possible worlds" that are consistent with the "world" specified in ρ.

Remark 1. *An important requirement for recovering a policy decision from a partial model ρ is that the decision dec should only be recoverable if it would have been the same one computed by all η in $\mathsf{Compl}\,(\rho)$.*

Ideally, an algorithm that can positively verify such "consensus" for $\mathsf{Compl}\,(\rho)$ should be efficient. And this may mean that the algorithm may at times not be able to detect such a consensus as there may be exponentially many maximal completions.

Example 1. *Kleene's 3-valued logic, detailed further below, is an under-approximation that is efficient and can verify such "consensus" but may also miss such "consensus" at times. For example if $\rho(x < 5 \cdot y) = \bot$, that logic will also evaluate $(x < 5 \cdot y) \vee \neg(x < 5 \cdot y)$ to $\bot$ even though all completions of ρ evaluate that condition to true.*

Because of the need for efficient algorithms this suggests that we will always have to face cases in which $\rho(cond)$ will evaluate to $\bot$.

That same requirement of Remark 1 applies to the computation of truth values for composed conditions $cond$: if $\eta(cond)$ is the same truth value v (either true or false) for all η in $\mathsf{Compl}\,(\rho)$, then $\rho(cond) = v$ is a safe definition that recovers incomplete information that ρ may contain. In fact, this "consensus" principle can also be applied to terms:

Example 2. *Consider a partial model ρ with $\rho(x) = \bot$. Then $\rho(x \cdot 0) = 0$ is a safe recovery of incomplete information since 0 is the result computed by all η in $\mathsf{Compl}\,(\rho)$, given that 0 is a zero of multiplication.*

Now consider a policy pol for which ρ evaluates at least one of its conditions to $\bot$. If there are completions of ρ that compute a different decision for pol, we will have to deal with incomplete information differently. For example, a PDP may then apply a policy wrapper or we may have a way of resolving incompleteness at the level of rules already, effectively casting $\bot$ into a truth value.

Before we discuss such approaches, let us review Kleene's 3-valued logic.

2 Kleene's Strong 3-Valued Logic

In that logic, the intuition is that true denotes the set $\{\mathsf{true}\}$, false denotes the set $\{\mathsf{false}\}$, and $\bot$ denotes the set $\{\mathsf{false}, \mathsf{true}\}$. Therefore, $\bot$ captures lack of knowledge as it may be either true or false.

We may then extend the semantics of propositional logic to these three values as follows. For any operator of that logic, e.g. conjunction, we apply it point-wise to all elements of its argument sets (which are one of the three sets above) and collect the results in an output set. For example:

$$\{\mathsf{false}, \mathsf{true}\} \;\&\&\; \{\mathsf{true}\} = \{\mathsf{false} \;\&\&\; \mathsf{true}, \mathsf{true} \;\&\&\; \mathsf{true}\} = \{\mathsf{false}, \mathsf{true}\}$$

gives us that $\bot \;\&\&\; \mathsf{true} = \bot$. In this manner, we can define the truth tables for Kleene's strong 3-valued semantics of propositional logic as depicted in Figure 1.

This semantics implicitly defines the computation of a truth value (true, false or $\bot$) for $\rho(cond)$ with respect to a partial model ρ:

1. We use the values of $\rho(t)$ for atomic terms t to compute the values of composite terms with respect to ρ.

x	$\neg x$
false	true
$\perp$	$\perp$
true	false

x && y		y false	y $\perp$	y true
	false	false	false	false
x	$\perp$	false	$\perp$	$\perp$
	true	false	$\perp$	true

$x \mid\mid y$		y false	y $\perp$	y true
	false	false	$\perp$	true
x	$\perp$	$\perp$	$\perp$	true
	true	true	true	true

Figure 1: Kleene's strong 3-valued propositional logic

2. With the values $\rho(t)$ for all terms, atomic and composite, we may then compute the truth values $\rho(atCond)$ for all atomic conditions $atCond$; these values are either true, false or $\perp$.
3. We then use Kleene's 3-valued semantics of propositional logic to compute the truth values of all composite conditions from the truth values of all atomic conditions.

Note that the first two steps may take advantage of some static rules, e.g. that $\perp \cdot 0 = 0$ or that $-\infty < \perp$ equals true.

3 Casting Incomplete Information Into Rules

One approach may be to cast $\perp$ into true or false in the application of rules, and to do this dependent on the type of rule.

For example, we may say that in the evaluation of `grant if` $cond$ under ρ for which $\rho(cond) = \perp$ under Kleene's strong 3-values semantics, we would interpret $\perp$ as false. This means that missing information prevents the grant based on that rule but only makes the rule not apply.

An interpretation of `deny if` $cond$ for $\rho(cond) = \perp$ may then be to cast $\perp$ into true, and so this rule would compute a `deny` on missing information of its condition.

This seems plausible: incomplete information for a grant rule is interpreted as lack of evidence for granting, but not as evidence for denying. Whereas incomplete information for a deny rule is interpreted as presence of evidence for denying, since otherwise an adversary could withhold such information in order to prevent a denial. An ability to withhold information to prevent a grant seems to be less of a concern, although it could be part of a denial of service attack.

Of course, such an approach needs to ensure that an attacker cannot exploit this approach so that the overall policy won't turn intended denials into grants or intended grants into denials. And policy analysis seems to be a means of getting such assurance.

Example 3. *For example, a "truth ordering" negation operator*

```
case {
  [pol eval grant: deny]
  [pol eval deny: grant]
  [true: pol]
}
```

would corrupt the intent of this approach. Let pol be a rule `deny if` $cond$*. Then an adversary could try to ensure that the PIP or PDP obtain incomplete information captured in a partial model ρ such that $\rho(cond) = \perp$. Under that model and the above approach of resolving truth value $\perp$ of a condition within rules, this would evaluate pol to* `deny` *and then the above negation operator would turn this* `deny` *into a* `grant`*.*

Of course, we may say that this is not a problem, since the policy writer was happy to negate policy outcomes in the truth ordering. But there still seems to be something wrong here since the overall, composed policy allows an adversary to manipulate the policy decision by withholding information. Naturally, we need to assume that the adversary knows exactly how policies compute meaning under incomplete information.

An important thing to realize is that there is probably no method that can resolve incomplete information and not expose itself to some denial of service. The aim seems to be to have a method in which such abilities are controlled, minimized, and analyzable.

4 Policy Analyses

Let pol be a FROST policy and ρ a partial or total model. We write $\rho \models pol\ \texttt{eval}\ dec$ to denote that under model ρ the policy pol does compute decision dec. Note that $\models$ implicitly refers to a mechanism by which $\rho(cond) = \bot$ instances get resolved in the policy evaluation, be it as discussed in the previous section or through some other method such as a top-level wrapper.

Suppose we are interested in whether an adversary could withhold information that would then change the policy decision from dec to some dec' where $dec \neq dec'$. For example, we may want to know whether this could change a `deny` outcome into a `grant`. We seem to be able to express this by existential quantification over partial models as follows:

$$\exists\, \rho, \rho' \colon (\rho \trianglelefteq \rho')\ \&\&\ (\rho \models pol\ \texttt{eval}\ dec)\ \&\&\ (\rho' \models pol\ \texttt{eval}\ dec') \tag{1}$$

For example, if dec is `grant` and dec' is `deny`, then ρ is the attack model that the adversary wants to realize and any of the completions of ρ' may be the intended model under which the policy should evaluate.

The adversary therefore needs to ensure that all t with $\rho'(t) \neq \bot$ get turned into $\bot$ values. Of course, we are interested in witnesses to the satisfiability of the formula in (1) for which ρ has a minimal number of $\bot$ values for atomic terms, as this makes it easier for the adversary to achieve this change in policy decision.

5 Evaluating Two Circuits Under Incomplete Information

Let us investigate whether there is a sensible way in which we can evaluate the circuits $\mathsf{GoC}(pol)$ and $\mathsf{DoC}(pol)$ under partial models ρ and still get conservative results.

Let us revisit the composition operator of Example 3. We write Q for this composition when its argument P equals $P = \texttt{grant if}\ cond$, and let Q' be that composition when P equals $\texttt{deny if}\ cond$, for which we write P' subsequently.

First, we compute the two Boolean circuits $\mathsf{GoC}(Q')$ and $\mathsf{DoC}(Q')$. We note that

$$\begin{aligned} \mathsf{T}(P'\ \texttt{eval grant}) &= \mathsf{false}\ \&\&\ \neg cond = \mathsf{false} \\ \mathsf{T}(P'\ \texttt{eval deny}) &= \neg\mathsf{false}\ \&\&\ cond = cond \end{aligned}$$

From this we can compute the first circuit as

$$\begin{aligned} \mathsf{GoC}(Q') = {} & \mathsf{false}\ \&\&\ \mathsf{false}\ ||\ \neg\mathsf{false}\ \&\&\ cond\ \&\&\ \mathsf{true}\ || \\ & \quad \neg\mathsf{false}\ \&\&\ \neg cond\ \&\&\ \mathsf{true}\ \&\&\ \mathsf{false} \\ = {} & cond \end{aligned}$$

Similarly, we compute

$$\begin{aligned}\mathsf{DoC}(Q') &= \mathsf{false}\ \&\&\ \mathsf{true}\ ||\ \neg\mathsf{false}\ \&\&\ cond\ \&\&\ \mathsf{false}\ ||\\ &\quad\ \neg\mathsf{false}\ \&\&\ \neg cond\ \&\&\ \mathsf{true}\ \&\&\ cond\\ &= \mathsf{false}\end{aligned}$$

Recall that both circuits encode the complete behavior of the policy:

- if both output true, the result is `conflict`,
- if both output false, the result is `undef`,
- if $\mathsf{DoC}(pol)$ outputs true and $\mathsf{GoC}(pol)$ outputs false, the result is `deny`, and
- if $\mathsf{DoC}(pol)$ outputs false and $\mathsf{GoC}(pol)$ outputs true, the result is `grant`.

It should be clear that this means that the above pair of circuits has the same meaning as the policy `grant if` $cond$, i.e. as policy P. These meanings are over all *maximal* models in which there is therefore no incomplete information.

Suppose that $\rho(cond) = \bot$ and so ρ is a partial model in which there is incomplete information. For this discussion, it does not matter whether $cond$ is an atomic or a composite condition; we only need that some incomplete information makes the evaluation of $cond$ under ρ return $\bot$.

If we take the approach that any `deny` rules with a condition $cond$ will then deny, this means that P' will evaluate to `deny` under ρ. But then Q' will evaluate to `grant` under ρ. This is the problematic case in which withholding information turns an intended conservative interpretation of lack of evidence for a `deny` as a denial into an actual `grant`.

Interestingly, the behavior of that approach on the policy `grant if` $cond$, which has the *same* meaning as Q' over all models with complete information, will render `undef` as result. One consequence of this example is that this approach does not preserve behavioral equivalences of policies over models with complete information.

5.1 A Possible Approach

Recall that any policy pol is equivalent, over all maximal models, to its `join` normal form

$$(\texttt{grant if } \mathsf{GoC}(pol))\ \texttt{join}\ (\texttt{deny if } \mathsf{DoC}(pol))$$

Definition 3 (Partial evaluation of Boolean circuits)**.** *Let ρ be a partial model. We may evaluate pol through this normal form and Kleene's strong 3-valued semantics as follows:*

1. *Evaluate $\mathsf{GoC}(pol)$ with Kleene's semantics to a truth value v_{GoC}, which is true, false or $\bot$.*
2. *Evaluate $\mathsf{DoC}(pol)$ with Kleene's semantics to a truth value v_{DoC}, which is true, false or $\bot$.*
3. *If v_{GoC} equals $\bot$, then we change the value of v_{GoC} to false.*
4. *If v_{DoC} equals $\bot$, then we change the value of v_{DoC} to true.*
5. *With these possibly overwritten values of v_{GoC} and v_{DoC}, we compute the policy result as follows:*
   - *if both v_{GoC} and v_{DoC} are true, return `conflict`,*
   - *if both v_{GoC} and v_{DoC} are false, return `undef`,*
   - *if v_{GoC} is true and v_{DoC} is false, return `grant`,*
   - *otherwise, return `deny`.*

Consider Q' above with $\rho(cond) = \bot$. Then v_{GoC} equals $\bot$ and gets overwritten into false. The value v_{DoC} is false and so the overall result is `undef` as intended. Similarly, consider Q with $\rho(cond) = \bot$. Then v_{GoC} is false and v_{DoC} is $\bot$ and gets overwritten to true. Therefore, the computed result is `deny` as intended.

Let us next say something about the type of theorems that we mean to be able to prove for any of the methods for evaluation policies for incomplete information, including for the one given in Definition 3. Let ρ be the partial model that is being evaluated. Then we want re-assurance that the evaluation under ρ' with $\rho' \trianglelefteq \rho$ does not allow for an undesired change of the result, e.g. from a `deny` to a `grant`.

Similarly, we may want a certain kind of monotonicity, meaning that the result computed for any ρ'' with $\rho \trianglelefteq \rho''$ is monotonically above the result computed for ρ. For example, for monotonicity with respect to the truth ordering of the Belnap lattice we would say that if the result is `grant` under ρ, then we may expect that the result is also `grant` for all ρ'' with $\rho \trianglelefteq \rho''$.

It is conceivable that both of these concerns, in which an attacker may either withhold information or supply more information, may be captured by monotonicity results.

For the above method of Definition 3 that appeals to the `join` normal form we can indeed prove such a theorem. Consider the table in Figure 2. This shows that the above method computes a decision that is the least (i.e. the meet) in the truth ordering of the Belnap lattice amongst those decisions that could in principle be computed if any $\perp$ outputs of these two Boolean circuits were to be refined into true or false.

GoC(pol)	DoC(pol)	computed decision	possible decisions
$\perp$	$\perp$	`deny`	`deny`, `conflict`, `undef`, `grant`
$\perp$	true	`deny`	`deny`, `conflict`
$\perp$	false	`undef`	`undef`, `grant`
true	$\perp$	`conflict`	`conflict`, `grant`
false	$\perp$	`deny`	`deny`, `undef`

Figure 2: Outputs computed as in Definition 3 of the two Boolean circuits where at least one output is $\perp$ (third column); what decisions are possible over complete models that refine the $\perp$ outputs into true or false (fourth column); in all rows, computed decisions are the meet in the truth ordering of the Belnap lattice over all possible decisions.

This at least confirms that this method is conservative in regard to refining incomplete models. Note that this also integrates well with having a top-level wrapper that converts any `conflict` or `undef` into `deny` – since the computed decision is never `grant` in situations in which at least one Boolean circuit outputs $\perp$ – as seen in Figure 2.

Let us now turn to the other issue, that an adversary may withhold information in order to influence the policy evaluation to his advantage. Consider that ρ is the partial model under which pol would be evaluated if the adversary did not have an opportunity to withhold information. Note that ρ may be a maximal model or may also have $\perp$ values, e.g. due to non-adversarial system faults.

An adversary who then withholds information that is present in ρ constructs a partial model ρ' with $\rho' \trianglelefteq \rho$ such that pol gets evaluated under ρ' and not under ρ. We mean to show that our method ensures that this can only decrease the decision in the truth ordering of the Belnap lattice.

Let us write $\trianglelefteq$ for the information ordering over the 3-valued logic as well, where $\perp \trianglelefteq$ true and $\perp \trianglelefteq$ false, and true and false are maximal elements in that partial order. It is easy to prove that $\rho' \trianglelefteq \rho$ implies that $\rho'(cond) \trianglelefteq \rho(cond)$ for all formulas of propositional logic $cond$.

In particular, if $\rho'(\text{GoC}(pol))$ and $\rho'(\text{DoC}(pol))$ are different from $\perp$, then $\rho(\text{GoC}(pol))$ equals $\rho'(\text{GoC}(pol))$ and $\rho(\text{DoC}(pol))$ equals $\rho'(\text{DoC}(pol))$, meaning that ρ' and ρ compute the same decision of policy pol.

Otherwise, at least one of $\rho'(\text{GoC}(pol))$ and $\rho'(\text{DoC}(pol))$ equals $\perp$ and then the computed decision for pol under ρ' is as specified in Figure 2. In particular, if ρ' then wants to realize a decision other than `deny` it can only be one of the following:

- `undef` in the third row: in that case ρ either computes the same results for the circuits (and so no attack occurs), ρ refines the sole $\perp$ to $\mathtt{true}$ (so the adversary turns a `grant` into an `undef`) or refines it to false (so the adversary cannot change the decision of `undef` at all).
- `conflict` in the fourth row: in that case ρ either computes the same results for the circuits (and so no attack occurs), ρ refines $\perp$ to $\mathtt{true}$ (and so the adversary cannot change the decision of `conflict` at all) or refines it to false (so the adversary would change a `grant` into a `conflict`).

This suggests that, apart from being able to realize a denial to a request, an adversary has basically no ability in manipulating policy outcomes to his advantage by withholding attribute information: for example, even though a change from `grant` to `conflict` might be a concern within a composition context, we evaluate the policy for the PDP as a composite and so this seems to be a non-issue.

Note that a deny-by-default wrapper for the table in Figure 2 would deny whenever one of the circuits outputs $\perp$. So one may wonder what is gained by the above approach in Definition 3 compared to denying whenever some attribute information is missing. The benefit of the above approach is that it can recover policy decisions other than `deny`, even `grant`, and can do this by also preventing the escalation of privileges (e.g. to turn an `undef` into a `grant`).

Reconsider the `join` normal form of a policy pol but with obligations obl_g^* for granting and obl_d^* for denying:

$$\big(\mathtt{grant}\,\{obl_g^*\}\ \mathtt{if}\ \mathsf{GoC}(pol)\big)\ \mathtt{join}\ \big(\mathtt{deny}\,\{obl_d^*\}\ \mathtt{if}\ \mathsf{DoC}(pol)\big)$$

After applying Definition 3 the computed set of obligations for the `join` normal form should be consistent with obligations from pol whenever pol is a $rule$ with obligations itself, i.e. either $pol = \mathtt{grant}\,\{obl_g^*\}\ \mathtt{if}\ cond$ or $pol = \mathtt{deny}\,\{obl_d^*\}\ \mathtt{if}\ cond$. Therefore, the computation of obligations under incomplete information can build upon the clauses of the original FROST language by slightly changing the compilation rules to reflect the 3-valued outcome of $\rho(cond)$.

Definition 4 (Obligations from partially evaluated Boolean circuits)**.** *Let ρ be a partial model. We may compute obligations for the partially evaluated policy through Kleene's strong 3-valued semantics as follows:*

$$\mathsf{oblg}(dec,\, \mathtt{grant}\,\{obl^*\}\ \mathtt{if}\ cond,\, \rho)\ \equiv\ \begin{cases}\{obl^*\} & \textit{if } dec = \mathtt{grant} \textit{ and } \rho(cond) = \mathsf{true}\\ \{\,\} & \textit{otherwise}\end{cases}$$

$$\mathsf{oblg}(dec,\, \mathtt{deny}\,\{obl^*\}\ \mathtt{if}\ cond,\, \rho)\ \equiv\ \begin{cases}\{obl^*\} & \textit{if } dec = \mathtt{deny} \textit{ and } \rho(cond) \in \{\mathsf{true},\, \perp\}\\ \{\,\} & \textit{otherwise}\end{cases}$$

$$\mathsf{oblg}(dec,\, \mathtt{case}\,\{\,[g_1 : p_1]\ \ldots\ [g_{n-1} : p_{n-1}]\,[\mathsf{true} \colon p_n]\,\},\, \rho)\ \equiv\ \mathsf{oblg}'(dec,\, g_i,\, \rho) \cup \mathsf{oblg}(dec,\, p_i,\, \rho)$$
$$\textit{where } \rho\left(\overline{\mathsf{R}(g_i)}\right) = \mathsf{true}$$

where $\overline{\mathsf{R}(guard)}$ denotes the evaluation of $\mathsf{R}(guard)$ under Definition 3.

That is to say, the truth value of guards pol `eval` dec is determined by computing the decision for pol as under Definition 3, and then we check equality of that result with dec in the usual 2-valued meaning of equations.

Consider Q' above with $\rho(cond) = \perp$. Here, we denote by $cond{\downarrow}$ and $cond{\uparrow}$ the conversion of the truth value of a condition $cond$ from $\perp$ to false and true, respectively. Then the obligations for granting and denying of Q' compute to

$$\mathsf{GObl}(Q') = \begin{cases} \mathsf{GObl}(P') & \text{if false} \\ \{\,\} & \text{if } cond\uparrow \\ \mathsf{GObl}(P') & \text{if } \neg(cond\uparrow) \end{cases} \qquad \mathsf{DObl}(Q') = \begin{cases} \{\,\} & \text{if false} \\ \mathsf{DObl}(P') & \text{if } cond\uparrow \\ \mathsf{DObl}(P') & \text{if } \neg(cond\uparrow) \end{cases}$$

and $\mathsf{GObl}(P') = \{\,\}$ which reflects that the only source of obligations is the `deny`-rule P'. These obligations are only issued if the decision for Q' were to deny, but the first arm of the `case`-clause can never logically be reached for given P' anyway and the second and third arms will never lead to a `deny`.

$\mathsf{GoC}(pol)$	$\mathsf{DoC}(pol)$	computed decision	$\mathsf{GObl}(pol)$	$\mathsf{DObl}(pol)$
$\perp$	$\perp$	`deny`	$\{\,\}$	$\{obl_d^*\}$
$\perp$	true	`deny`	$\{\,\}$	$\{obl_d^*\}$
$\perp$	false	`undef`	$\{\,\}$	$\{\,\}$
true	$\perp$	`conflict`	$\{\,\}$	$\{\,\}$
false	$\perp$	`deny`	$\{\,\}$	$\{obl_d^*\}$

Figure 3: Outputs computed as in Definition 4 of the two Boolean circuits where at least one output is $\perp$ (third column) and which obligations follow from the computed decisions (fourth and fifth column).