One-Way Model Transformations in the Context of the Technology-Roadmapping Tool IRIS

Florian Sihler, Matthias Tichy, and Jakob Pietron
Institute of Software Engineering and Programming Languages, Ulm University, Germany

ABSTRACT CONTEXT: Today's industry heavily depends on the collaboration of multiple companies that have to exchange a lot of information. When these companies use technology roadmapping and model-driven development, this leads to the exchange of very large models, which is governed by competition laws, and the companies' interest in protecting their intellectual property. OBJECTIVE: We suggest an automated one-way model transformation approach to remove content worth protecting from a given model in the context of the graphical technology-roadmapping modeling tool IRIS. METHOD: We elaborate constraints that an obfuscated model must fulfill. Based on these, we systematically identify and analyze appropriate one-way transformations to enable the desired level of obfuscation, implementing a proof of concept implementation in IRIS. Our approach first transforms the model into a flattened constraint system and uses nine selected transformations—ranging from constant folding to the tailoring of expressions—to create a new model without the sensitive content. We evaluate our transformations according to a set of predefined properties and constraints regarding their effects on the behavior of the model and the inferable information. LIMITATIONS: Our obfuscation concerns the model and the embedded formulas but does not deal with individual data, which may be subject to privacy-related issues. RESULTS: We find that the identified transformations are applicable in practice and that they can remove a lot of sensitive information from a given model. Future work includes dealing with individual data and protecting against reverse-engineering attacks by domain experts.

KEYWORDS Model Transformation, Model Obfuscation, Technology Roadmapping, Compliance, IP-Protection.

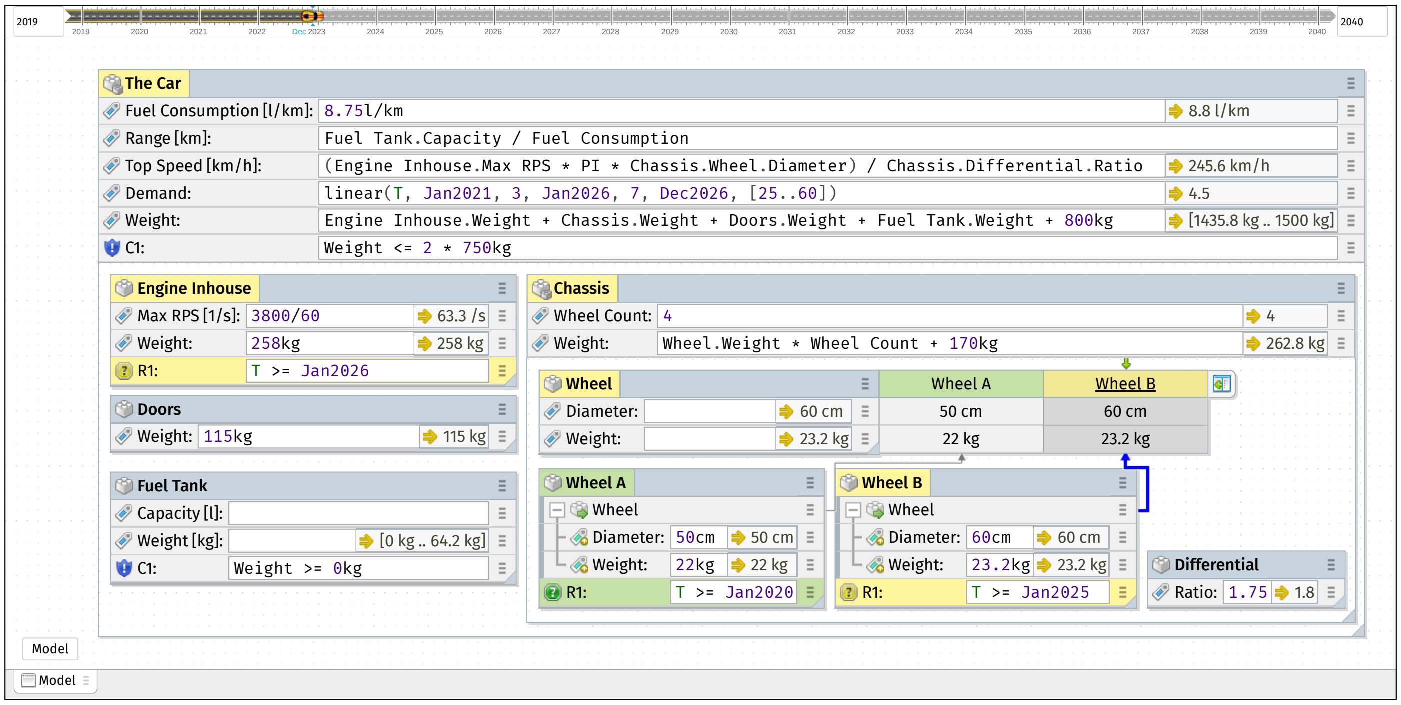
1. Introduction

Value chains are one of the fundamental concepts in modern businesses, often implemented through technology roadmaps. Those roadmaps enable collaboration and help to identify industry needs, granting invaluable insights for businesses (Garcia 1997). They can be used to produce models that represent and forecast the development and structure of innovation (Rinne 2004). Because of their omnipresence, there are a lot of different formats and types of technology roadmaps (Phaal et al. 2001; Garcia & Bray 1997).

Although many innovations require the collaboration of multiple parties, there are many reasons against distributing the complete model to everyone involved. Those include (*a*) simplifications or complexity reductions so that the receiving party can grasp the model more easily from its perspective, (*b*) the removal of content that is worth protecting such as trade secrets and intellectual property (IP), or (*c*) staying compliant with various competition laws (BusinessEurope 2017). As a result, every party only owns and knows pieces of one full model, which may never be viewed or stored as a whole by anyone.

Nevertheless, these parties have to exchange information when working together. As an example, consider a fuel tank manufacturer that needs information about the upper weight limit for a fuel tank—which may change over time due to innovations—and, in return, an automotive OEM who has to know the capacity of the fuel tank proposed. The state of the art extracts the (relevant) parts by hand (e.g., in Microsoft Excel, Microsoft Word, or Microsoft Visio). Afterward, the extracted parts are sent to the fuel tank manufacturer, who in turn has to

JOT reference format:
Florian Sihler, Matthias Tichy, and Jakob Pietron. *One-Way Model Transformations in the Context of the Technology-Roadmapping Tool IRIS.* Journal of Object Technology. Vol. 22, No. 2, 2023. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2023.22.2.a2


2019 2040

The Car
Fuel Consumption [l/km]: `8.75l/km` → 8.8 l/km
Range [km]: `Fuel Tank.Capacity / Fuel Consumption`
Top Speed [km/h]: `(Engine Inhouse.Max RPS * PI * Chassis.Wheel.Diameter) / Chassis.Differential.Ratio` → 245.6 km/h
Demand: `linear(T, Jan2021, 3, Jan2026, 7, Dec2026, [25..60])` → 4.5
Weight: `Engine Inhouse.Weight + Chassis.Weight + Doors.Weight + Fuel Tank.Weight + 800kg` → [1435.8 kg .. 1500 kg]
C1: `Weight <= 2 * 750kg`

Engine Inhouse
Max RPS [1/s]: `3800/60` → 63.3 /s
Weight: `258kg` → 258 kg
R1: `T >= Jan2026`

Doors
Weight: `115kg` → 115 kg

Fuel Tank
Capacity [l]:
Weight [kg]: → [0 kg .. 64.2 kg]
C1: `Weight >= 0kg`

Chassis
Wheel Count: `4` → 4
Weight: `Wheel.Weight * Wheel Count + 170kg` → 262.8 kg

Wheel		Wheel A	Wheel B
Diameter:	→ 60 cm	50 cm	60 cm
Weight:	→ 23.2 kg	22 kg	23.2 kg

Wheel A
Wheel
Diameter: `50cm` → 50 cm
Weight: `22kg` → 22 kg
R1: `T >= Jan2020`

Wheel B
Wheel
Diameter: `60cm` → 60 cm
Weight: `23.2kg` → 23.2 kg
R1: `T >= Jan2025`

Differential
Ratio: `1.75` → 1.8

Model

Figure 1 A small model of a car in IRIS. It showcases the perspective of an OEM which needs a fuel tank (represented by the "Fuel Tank" block) by a fuel tank manufacturer. While the OEM has certain constraints the fuel tank has to fulfill—the "Weight" property must be in the range of 0 kg to 64.2 kg—due to the embedded intellectual property the OEM can not share the whole model with the fuel tank manufacturer. When selecting "Fuel Tank" to be obfuscated, we must remove all other elements but keep their impact on "Fuel Tank" intact (i.e., the inferred range of "Fuel Tank.Weight"). Figure 3 shows the resulting obfuscated model.

repeat the process to send a model back. This is a tedious and impractical process. Moreover, with a lot of parties, communication, and changes, the manual (re-)creation of models for specific partners is costly and error-prone, increasing the risk of leaking unwanted or unauthorized information (Debreceni 2019).

Accordingly, there is the need for an automated model transformation process. Given a model and a selection of parts of the model, this process should produce an exported model that contains the selected parts. Additionally, these parts should behave equivalent to their counterpart in the original model (i.e., they should evaluate to the same results), while revealing as little information as possible about the original model (Sihler 2023). To address this, we formulate one research question:

RQ1: What are necessary and adequate model transformations that remove unselected parts from the model, but keep the behavior of the selected parts?

The main contribution of this paper is the selection and implementation of nine transformations as part of an automated four-phase transformation process in the context of the technology roadmapping tool IRIS. These transformations answer RQ1 and aim to satisfy four constraints which we derived together with our industry partners. While we introduce all of them in the context of IRIS, we exemplify their generalizability for other use cases like the one-way transformation of Excel spreadsheets which are omnipresent in the industry.

The remainder of this paper is as follows: with Section 2 as an introduction to IRIS, the problem definition in Section 3 presents a set of four constraints our transformations must fulfill, and Section 4 differentiates our approach from the related work. In Section 5 we present all nine selected transformations to address RQ1, evaluate the approach in Section 6, and discuss its limitations and its generalizability in Section 7. Section 8 summarizes our contribution and considers potential future work.

2. Running Example with IRIS

IRIS is a graphical modeling tool for technology roadmaps (TRM). Compared to other tools, IRIS not only supports domain experts in modeling future innovative systems, their structure, properties, and requirements but it also supports them in deriving the need for innovation (Pietron et al. 2022), e.g., if a component is not available at a certain point in time or a component can not fulfill the given requirements in the future. In this case, IRIS allows to narrow down properties a solution must have. IRIS treats time, time-dependency, and availability as a first-class citizen and allows users to model uncertainty, with its Excel-like domain-specific expression language supporting intervals, ternary logic, types, units, and references to other model elements. A built-in solver solves all equations and evaluates the fulfillment of requirements.

We use IRIS as the basis for our one-way model transformation process. Hence, it is necessary to understand (*i*) how IRIS represent models, and (*ii*) how the IRIS solver evaluates the model. While Breckel et al. (2021) offer an in-depth discussion

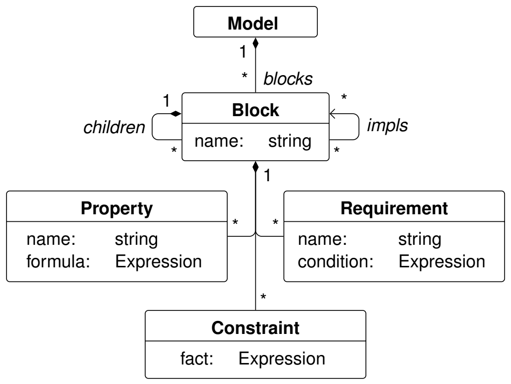


Figure 2 Simplified metamodel of the IRIS-Language with blocks, properties, requirements, and constraints as model elements.

of the domain-specific language of IRIS and the behavior of its solver, this short overview focuses on the concepts relevant to understand the following sections.

Figure 1 presents a small model of an OEM loaded in IRIS. This model consists of a total of nine so-called "blocks" like "The Car" (identified by the icon) which can be used to represent products, concepts, or innovation in the modeling domain. In this example, the OEM modeled a specific car, only missing a concrete fuel tank which is to be produced by an external fuel tank manufacturer.[1]

The very top of Figure 1 consists of a time-slider—with currently located at December 2022—that can be used to configure the currently simulated "global time" of the model state. This time is represented by the symbol `T` within expressions. These expressions are linked to model elements that, in turn, are part of a block (cf. Figure 2). We differentiate three different types of model elements aside from blocks:

- properties, like "Weight" in the "Chassis" block, have a name ("Weight") and a value (shown to the right of : `262.8 kg`) described by an expression (`Wheel.Weight * Wheel Count + 170kg`). They can be referenced by their name in other expressions, with the blocks they are contained in acting as scopes. The value can be set explicitly or derived by the solver due to one or many given constraints.
- constraints have an expression that has to be fulfilled. For example, "C1" in the block "The Car" restricts the upper value of the "Weight" property to `1500kg`. Due to "C1", the solver can evaluate the "Weight" of the "Fuel Tank" to be in the range of 0 to 64.2 kg.
- requirements have a formula that is used to determine if the block that contains them is available at the currently simulated time (i.e., if the corresponding concept is available). "R1" in "Wheel B" is only fulfilled if the currently simulated time is at least January 2025 (`T >= Jan2025`).

Blocks themselves can have two special types of relationships. They can be *children* of a parent block—the blocks "Engine Inhouse", "Chassis", "Doors", and "Fuel Tank" are children of "The Car"—or *implementation alternatives* of a solution space, like "Wheel A" and "Wheel B" of "Wheel". Modeling a block and letting implementation alternatives derive from that block creates a solution space, which allows to compare the different alternatives. "Wheel B" is the currently the manually selected implementation in Figure 1. "Wheel" can be seen as an interface. It consists of all required information and requirements the OEM needs to build a (future) system. Consequently, such an interface is suitable to be exchanged with suppliers which vice versa can provide modeled implementations of that interface containing all relevant information. These implementations, again, can be added to the solution space to be compared and evaluated by the OEM.

Furthermore, blocks have implicit properties, with the most important one being an "Availability" property that is automatically created based on all of the block's requirements and the requirements of its children. The value of this property is visualized by the color of the background of the block's name. As an example, "Wheel A" is currently available (with `T >= Jan 2020`) as indicated by the green color, while "Engine Inhouse" will become available in the future (January 2026) as indicated by the yellow background. Blocks without requirements affecting them do not have a special background.

Solving To infer correct values for all properties and requirements, IRIS uses a built-in solver which flattens the model into a constraint system in the background, using unique IDs for each model element and by repeatedly applying symbolic transformations—as explained in further detail by Breckel et al. (2021). For a property like "Demand" in "The Car" this produces the constraint: `Demand(T) = linear(T, Jan2021, 3, Jan2026, 7, Dec2026, [25..60])` (with *linear* interpolating linearly between the value `3` for January 2021, `7` for January 2026, and the interval `[25..60]` for December 2026). The symbol "`T`" is added as an implicit parameter for each property and defaults to the currently simulated time if not set explicitly when referencing the name of a property.

3. Problem Definition

Based on requirements from our industry partners, we formalize the problem as follows: Given an IRIS-model A with its set of model elements $\mathbb{E}_A$ and a selection of those elements $S \subseteq \mathbb{E}_A$, we want to create another IRIS-model B which satisfies the four following constraints that we consider important to ensure that B no longer contains any content worth protecting while keeping the behavior of the selection intact:

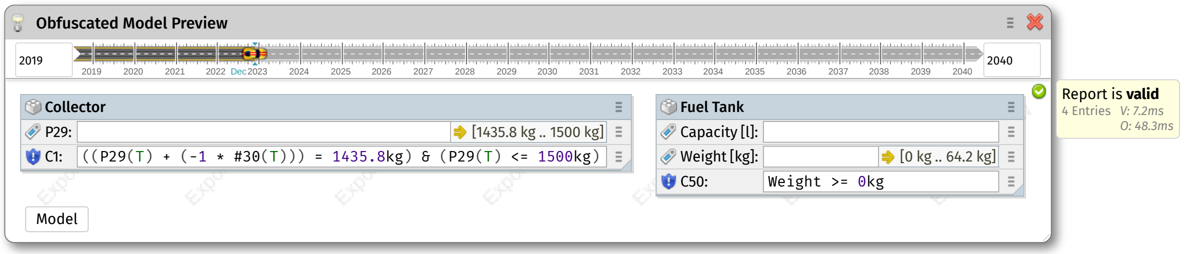
C1) *Self-Containment.* B has to be self-contained (i.e., it contains no reference to any model element $a \in \mathbb{E}_A$).

C2) *Selected.* B contains all $e \in S$ and none of those in $\mathbb{E}_A \setminus S$.

C3) *Inference.* B allows no inference on any $e \in \mathbb{E}_A \setminus S$. That is, B should reveal as little information as possible regarding the formulas or identifiers of any $e \in \mathbb{E}_A \setminus S$.

C4) *Behavior.* All selected elements, in models A and B, have to evaluate to the same value for every possible global time (i.e., they have to be "behavioral equivalent" as explained in Section 3.1).

[1] The example is available online at: https://tinyurl.com/iris-ecmfa23


Obfuscated Model Preview
2019
2019 2020 2021 2022 Dec2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040
2040
Model

Collector

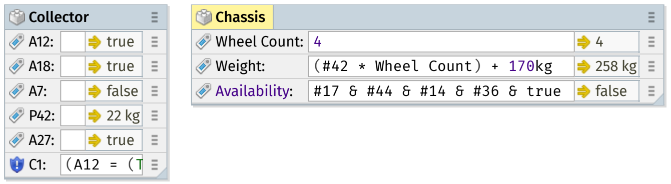
P29:		[1435.8 kg .. 1500 kg]
C1:	((P29(T) + (-1 * #30(T))) = 1435.8kg) & (P29(T) <= 1500kg)	

Fuel Tank

Capacity [l]:		
Weight [kg]:		[0 kg .. 64.2 kg]
C50:	Weight >= 0kg	

Report is **valid**
4 Entries *V: 7.2ms*
O: 48.3ms

Figure 3 Result of obfuscating the “Fuel Tank” block in Figure 1. While the “Collector” block, which we need to express the remaining effect of unselected elements we were unable to fully obfuscate, is normally hidden, it is shown for completeness. The ✓ icon shows that C1, C2, and C4 are satisfied (cf. the fourth phase in Section 5). If a modification by the user produces an error (e.g., by invalidating C3), the icon changes to ✗, problematic elements get highlighted in red and the hover-over of the icon (currently reading “Report is valid”) gives more information regarding the problem. The example is intentionally kept simple to allow manual verification of the produced result. See Figure 4 for another example.



Collector

A12:		true
A18:		true
A7:		false
P42:		22 kg
A27:		true
C1:	(A12 = (T	

Chassis

Wheel Count:	4	4
Weight:	(#42 * Wheel Count) + 170kg	258 kg
Availability:	#17 & #44 & #14 & #36 & true	false

Figure 4 Result of obfuscating the “Chassis” block in Figure 1 (without the window shown in Figure 3). The model alongside other examples can be explored at: https://tinyurl.com/iris-ecmfa23-obf-chassis.

Dealing with each of these constraints individually is a trivial task. For C1 it would be sufficient to create a copy of A, for C2 to restrict this copy to all $e \in S$. Similarly, just solving two or three of these constraints at a time is an obvious task. For example, to satisfy C1, C2, and C3 we can simply copy the $e \in S$ from A and remove all references to elements in $a \in \mathbb{E}_A$ in the expressions of all $e \in S$. This way we have a self-contained model, that contains all $e \in S$ without any mention of any model element $e \in \mathbb{E}_A \setminus S$.

However, trying to satisfy all constraints C1–C4 elevates the problem from “almost trivial” to impossible realms.We have to retain aspects from A which are not selected but relevant for the behavior of selected elements, without including any critical information about their origin and without embedding them directly, while keeping model B self-contained.[2] For this, we may need additional model elements in B, which summarize the calculations of the unselected elements so that the results of all $e \in S$ remain the same. In Figure 3, we need the “P29” property and the “C1” constraint in the “Collector” block, to ensure that the allowed weight range for “Fuel Tank.Weight” remains unchanged.

The resulting model shown in Figure 3 satisfies C1 (*Self-Containment*) as it no longer holds any reference to the original model shown in Figure 1. Furthermore, it satisfies C2 (*Selected*) with none of the unselected elements (“Wheel”, “Wheel.Diameter”, ...), but all of the selected (“Fuel Tank”, “Fuel Tank.Capacity”, ...) included. Regarding C3, the resulting model reveals rather little information about the original model. There are no wheels and, hence, no information regarding what kind of wheel is used. The car itself and information about all other blocks are missing, and the formulas like “The Car.Weight” have been massively reduced. We can not even (definitively) reason, what numbers like `1500kg` mean, as we only know, that this weight affects the weight of our fuel tank. In consequence, we consider C3 to be satisfied. Finally, all remaining elements (most notably “Fuel Tank.Weight”) behave exactly as in the original model, which satisfies C4 (*Behavior*).

With this, we structured the problem into four constraints that must hold for the output model. We use these constraints to analyze the selected transformations in Section 5.

[2] Because this is trivial for $S = \mathbb{E}_A$ and $S = \emptyset$, by either creating B as a full copy of A or by creating B as an empty model, we focus on cases where $S \neq \mathbb{E}_A$ and $S \neq \emptyset$.

3.1. Behavioral Equivalence

To compare the initial model with its obfuscated counterpart we use the concept of “behavioral equivalence” which is required by C4 (*Behavior*). Regarding the comparison of models, the authors Yücesan & Schruben (1992) present various definitions for sensible equivalence constructs based on simulation graph models. Within this paper, we define the behavior of a model resembling their definition of “input/output behavior”. This behavior includes the values to which model elements evaluate, the availability of specific components, and more. In short, every behavior that is perceivable by the user for a given input.

We restrict the term of behavioral equivalence to model elements both Iris-models A and B have in common—which allows both models to define additional elements to influence their behavior—which we denote as the intersection of their elements: $\mathbb{E}_A \cap \mathbb{E}_B$. Therefore, with T being the set of all allowed simulated global times, we define two A and B as “behavioral equivalent” if:

$$\forall t \in T \; \forall e \in \mathbb{E}_A \cap \mathbb{E}_B : \; e_A(t) = e_B(t).$$

$e_A(t)$ and $e_B(t)$ refer to the common model-element e from the models A and B, evaluated for the current point in time t, respectively. The behavioral equivalence is always trivially fulfilled for two models that have a disjoint set of model elements.

3.2. Disclaimer

Considering C3 (*Inference*), we focus on obfuscating the model alongside all contained formulas but our approach ignores individual data. That is, we do not consider the scramble of employee names in formulas—as it is done by "ARX" (Prasser et al. 2014)—or the distortion of integer values— done by AnonymousXL (Van Veen & Hermans 2014). This is due to the amount of existing research on this topic in the area of privacy-preserving data mining (Bakken et al. 2004).

4. Related Work

While there exists a lot of research on related problems, we are not aware of any research dealing with the very same problem as described in Section 3.

We read several extensive literature surveys and systematic literature reviews in the area of obfuscation and extracted over 100 surveys, papers, and technical reports (Balakrishnan & Schulze 2005; Hosseinzadeh et al. 2018; Ab. Rahim & Whittle 2015; Zhou et al. 2008). However, following up with papers mentioned in those sources and their related work, most of them focus on the obfuscation of complete programs (i.e., have no concept of a selection to be retained) or the removal of personal information from large datasets. Furthermore, many obfuscations and anonymization techniques do not require the output to be readable or in the same format. They either only retain metadata required for subsequent analysis or use an assembler or binary format for representation (Hosseinzadeh et al. 2018). Those operating differently almost exclusively use a single trusted entity or a set of multiple parties that receive extracted parts of a program or model to answer requests in an oracle fashion. However, none of these approaches yield a standalone program. Instead, they require online work to communicate with the other parties.

We list a subset of those closely related to one-way model transformations so that they cover a broad range of perspectives. We have loosely divided the related work into two groups: (*i*) model obfuscation, and (*ii*) obfuscation of code.

4.1. Model Obfuscation

Research in this field deals with the uni- and bidirectional transformation of one or several models into one or several models of a comparable abstraction level (Czarnecki & Helsen 2006).

Goettelmann et al. (2015) propose an approach to obfuscate business process models (BPM) at design time. In their paper, they present a block-based fragmentation of BPM using obfuscation constraints. While their goal resembles ours as they want to avoid, that a single cloud provider used by the company can understand critical fragments of a BPM, their way of achieving this goal differs as they do not obfuscate any contents or remove any information from the whole model. Instead, they split a single BPM into several fragments that can be distributed to different cloud providers which have to communicate with each other during the execution. Therefore, any fragment sent to a cloud service provider or another company requires communication with other model parts to be executed and to behave equivalently. This contradicts C1 which requires the model to be self-contained so it works without this communication with other parties. Additionally, there is no comparable "selection" mechanism (making C2 and C3 not applicable).

The goal of Gupta et al. (2017) is to protect CAD models against counterfeiting to protect intellectual property. However, they introduce additional spheres into the CAD models that do not dilute or obfuscate the present information but rather "sabotage" reproductions based on this model. Hence, most of the information of the original model remains but changes the "behavior" of that model when reproducing (contradicting C4 that wants to keep the behavior of selected parts unchanged).

Fill (2012) has the same goal of supporting IP-preserving collaboration in the context of models. Although he focuses on conceptual models (Karagiannis & Kühn 2002), a lot of transformations he describes are adaptable to the context of our problem definition (like "information hiding" which removes information from the model). However, he remains on an abstract level presenting abstract examples of what could be done but not when to apply or how to perform such transformations automatically, or what problems their application implies.

Functional Mockup Interfaces (Blockwitz et al. 2012) are similar to our approach in that they allow to define FMUs which serve as a black box that can be used by others without knowing the details of the implementation. However, FMUs (by default) still contain most of the formulas, although the compiler has obfuscated parts of them automatically by applying compiler optimizations (Cooper & Torczon 2011). Still, we could use our approach to generate an FMU as output.

4.2. Code related obfuscations

Code-related obfuscations do not deal with models or static data in tables (e.g., inside an Excel spreadsheet) but with code. The acquired methods make use of access restriction, control flow, and information obfuscation (Balakrishnan & Schulze 2005).

The authors Debreceni, Bergmann, Ráth, & Varró describe the insufficient administrative capabilities of source code repositories and introduce their MONDO-framework for rule-based access control policies (Debreceni et al. 2017, 2018). They essentially provide views of a source code repository that reveal only parts of the whole underlying model so that several crucial or secret aspects of a code are not revealed to industry partners.

However, their approach differs on several crucial points. First of all, they assume the existence of a so-called "gold model" that contains *all* of the information. Additionally, they describe the "front models" as mere secure views on this gold model and not as standalone and independent versions of it. That contradicts the constraint C1, requiring the one-way transformed model to be self-contained (cf. Section 3). While views of the model can be materialized (Preuß 2016) to achieve a similar effect, materialization would require the creation and adaption of similar transformations. Moreover, we assume that a "gold model" does not exist. Furthermore, their concept of a "lens" as a bidirectional model transformation is not guided by a selection of elements but by a collection of access rules (Bergmann et al. 2016).

While compilers usually do not obfuscate the code they compile, they use several transformations that aid our one-way model transformation purpose (Cooper & Torczon 2011). Especially optimizations that remove unreachable or dead code and that already pre-evaluate parts of a program. They hide calculations and remove origin information by inlining constant values. For example, we adapt the well-known compiler optimizations of constant folding and constant propagation and introduce them as T1 Arithmetic Simplification and T2 Inline Definitions in the following Section 5.

Sosonkin et al. (2003) focus on a subset of program obfuscations by hiding the design intents embedded in object-oriented applications. They present three transformations called (*i*) class coalescing, (*ii*) class splitting, and (*iii*) type hiding. Furthermore, they provide a design obfuscator for Java using all three obfuscation techniques. Even though the formalized models we use do not have the concept of classes, similar transformations can be applied to the structure. For example, by splitting a block in two or merging two blocks (cf. Section 8).

5. One-Way Model Transformations

To solve the problem as formulated in the problem definition in Section 3, we propose a one-way model transformation program that is split up into four distinct and sequentially executed phases. They are as follows (cf. Figure 5):

Phase 1: Normalize the input model and selection.
Phase 2: Prepare and populate supporting data structures.
Phase 3: Obfuscate by repeatedly applying transformations.
Phase 4: Verify the produced result.

First Phase – Normalization The first phase is closely related to the implementation domain—which is IRIS in our case—as it must know the syntax and semantics (like the automatically generated Availability-properties) of the modeling language. It transforms the input IRIS-model into a (finite) flattened constraint system, marking every constraint generated from a model element that is part of the selection. To achieve this phase, we make use of the IRIS-solver which flattens the model for us, as described in detail by Breckel et al. (2021).

Second Phase – Preparation The preparation phase is responsible for the initialization of all data structures necessary for the obfuscation phase. The most important part is its reference tracing—which creates a graph linking all references (e.g., names of properties) within the constraints—and its tracing model, which is used to trace all changes by the transformations of the next phase.

Third Phase – Obfuscation This is the main phase of the obfuscation process and described in detail in Section 5.2. Given the constraint system of the first phase and the additional data from the second phase, it (*i*) obfuscates the constraint system (C2, C3, C4), and (*ii*) creates a new model based on the obfuscated constraint system (C1) by the repeated application of nine transformations. These transformations work in four obfuscation subphases:

- *3.1 Outer*: Works only on unselected constraints.
- *3.2 Inner*: Works on the effects of unselected on selected constraints.
- *3.3 Structure*: Works on the hierarchical structure (e.g., the block relationships) and builds the output model.
- *3.4 Cleanup*: Removes redundant information and refines the models' appearance.

The main goal of the obfuscation phase is to use the input model and the selection to create the output model satisfying C1–C4.

Transformation	**Deterministic**	**Reversible**	**Global**	**Local**	**Phase**
T1 Arithmetic Simplification	●			●	1, 2
T2 Inline Definitions	●	⊙	●	⊙	1, 2
T3 Expression Normalization	●	●		●	1, 2
T4 Tailor Expressions	⊙		●		1
T5 Reduce Constants	●		●		1, 2
T6 Identifier Obfuscation	⊙		●		3
T7 Remove Orphaned	⊙		●		3
T8 Rebuild Hierarchy	⊙		●		3
T9 Remove Implicit	⊙	●	●	⊙	4

Table 1 Relevant properties of all nine transformations. (●: Required/Fulfilled, ⊙: Can exploit/fulfill, *Default*: Does *not* require/fulfill).

Fourth Phase – Verification After the model is obfuscated, a domain expert may want to conduct manual changes (e.g., change the layout of the model or some expressions). In general, we allow for any change to the obfuscated model which allows the user to (accidentally) violate any of the constraints. Therefore, we use the obfuscated model (with potential changes by the user), the initial model, and a mapping of the selected elements in both models, to continuously and automatically verify the constraints C1, C2, and C4.

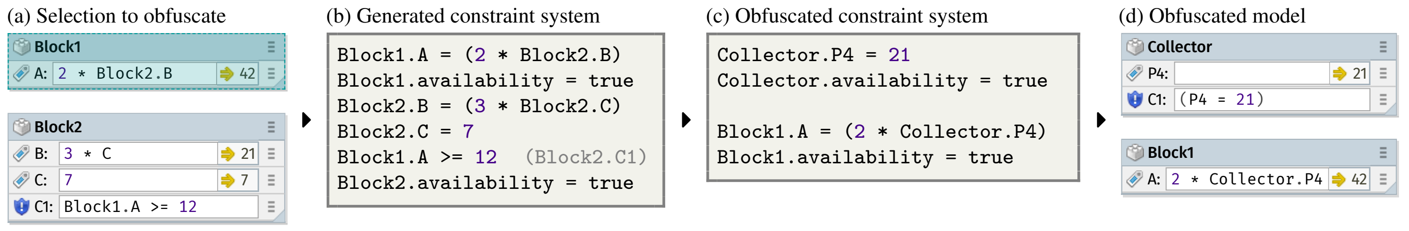
For the behavioral equivalence, we have implemented a checking function that is restricted (*i*) to the time interval defined by the time slider (in Figure 1 this would be the range from 2019 to 2040), and (*ii*) only checking for discrete time steps (e.g., on a monthly basis). While this is not the same as proving that both models are behavioral equivalent (as IRIS uses a floating point number to represent time), this implementation seems to be sufficient for the applications of our industrial partners (because in their technical roadmaps they plan and operate on a time scale of months).

5.1. Transformation Properties

We recognize a set of properties to be analyzed for each transformation individually. These properties are based on the works of Amrani, Dingel, et al. (2012) and Collberg et al. (1997).

Deterministic: A transformation is considered to be deterministic if it always performs the same modifications for the same input.

Reversible: A transformation is "reversible" if its modification can be reversed without any or with relatively little guessing. Alternatively, we consider it reversible as well if it


(a) Selection to obfuscate

```
Block1
A: 2 * Block2.B   42

Block2
B: 3 * C   21
C: 7   7
C1: Block1.A >= 12
```

(b) Generated constraint system

```
Block1.A = (2 * Block2.B)
Block1.availability = true
Block2.B = (3 * Block2.C)
Block2.C = 7
Block1.A >= 12  (Block2.C1)
Block2.availability = true
```

(c) Obfuscated constraint system

```
Collector.P4 = 21
Collector.availability = true

Block1.A = (2 * Collector.P4)
Block1.availability = true
```

(d) Obfuscated model

```
Collector
P4:   21
C1: (P4 = 21)

Block1
A: 2 * Collector.P4   42
```

Figure 5 Phases of the one-way model transformations. With "Block1" selected in (a), (b) shows the constraint system generated by the first phase (simplified for this example). Afterward, the second phase traces all references (e.g., recognizes all locations of "Block1.A"). The third phase generates the new constraint system shown in (c) and builds the model shown in (d). The automatic verification of the fourth phase is visualized in Figure 3.

does not have to be reversed at all to grasp the meaning of the source (e.g., a reordering of $b + a$ to $a + b$). We use the property to mark transformations that alone are insufficient to obfuscate the model but serve different purposes (like allowing other transformations to function).

Our definition of the term "reversible" is a weaker version of "resilience" as defined by Collberg et al. (1997).

Scope: The scope categorizes the specific subphase(s) (cf. the third phase in Section 5) in which the transformation is to be applied: 1) outer, 2) inner, 3) structure, and 4) cleanup represented by the numbers 1 to 4 respectively.

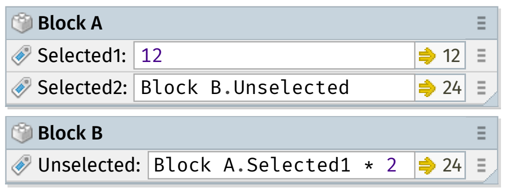
Additionally, it specifies if the application of the transformation is *Local* (it operates on a single constraint) or *Global* (it modifies multiple constraints, or requires information on/from other constraints).

5.2. Transformation Catalog

From an abstract perspective, all transformations considered are to be executed in the obfuscation phase and are applied according to the execution order proposed in Section 5.3. They form the basis of the model-to-model approach to deal with the problem definition of Section 3. We consider two kinds of transformations: (*i*) those that change the constraints created with the normalization phase (T/1 – T/7), and (*ii*) those that create and modify the structure of the output model (T/8 , T/9).

Table 1 lists all transformations that are implemented in IRIS and discussed in this section, alongside a summary of their properties as described in Section 5.1. After discussing each transformation individually, we describe how they work together in Section 5.4. We have selected these transformations from a pool of transformations based on the following criteria:

- *Necessity:* Some transformations are required to achieve certain obfuscations. For example, if we want to obfuscate the hierarchy of a model, we need a transformation that modifies the model hierarchy (like T8 Rebuild Hierarchy).
- *Intuition:* Some transformations, as the evaluation of constants, are intuitive for a one-way transformation approach. If we evaluate $-2 \cdot 7.5$ to -15, it is impossible to recover the original formula as there are infinitely many terms that evaluate to -15.



```
Block A
Selected1: 12   12
Selected2: Block B.Unselected   24

Block B
Unselected: Block A.Selected1 * 2   24
```

Figure 6 Showcase of potential problems when executing T1 Arithmetic Simplification on functions with references. Here, the evaluation of `Block A.Selected1 * 2` which would break the link between `Selected1` and `Selected2` (assuming only the latter two are part of the selection).

5.2.1. *Disguise of Computing Paths*

Transformations of this group hide details about the computation of values. Therefore, for an inferred result of x, these transformations obfuscate parts of the required calculations to receive x. The simplest form of this obfuscation is evaluating the equations beforehand as performed by T1 Arithmetic Simplification.

T1 Arithmetic Simplification Essentially, this transformation performs *constant folding* (Cooper & Torczon 2011, Chp. 8) by evaluating functions when all of their arguments are free of any references (like names of other properties and blocks). For example, this transforms `2 + 5` to `7` and `ceil([1.3..2.7])` to `[2..3]` with "+" and "`ceil`" as the functions which are simplified (with "+" written in infix-notation).

To perform this transformation, it is enough to look at the constraint that is to be simplified. Therefore, we consider it to be *local*. Furthermore, because all functions in IRIS are deterministic, this transformation is *deterministic as well (as long as we evaluate every applicable function)*.

The following points discuss the influence of T1 Arithmetic Simplification on all four constraints from Section 3:

C1) *Self-Containment.* Functions in IRIS can not evaluate to expressions that include other references. Even if they could, we essentially break all references to A (by constructing a new model based on the transformed constraints in the *structure* obfuscation subphase). Therefore, this transformation is (by construction) unable to produce any reference to $a \in \mathbb{E}_A$.

C2) *Selected.* This transformation does only operate on a single constraint at a time. Furthermore, it does not perform any structural modification (restricting itself to the evaluation of expressions). Therefore, it can not alter the elements that are contained in B.

C3) *Inference*. In general, showing that evaluating a function does not reveal more about the model than the function itself, is hard. Consider a hypothetical function "`dump()`" that produces a human-readable string containing the original model. While the result may be removed (cf. T7 Remove Orphaned) if the result of `dump()` is not required by any selected constraint, we can not future-proof this transformation against all functions potentially added to Iris. For now, none of the functions provided by the domain-specific expression language of Iris have a problematic behavior similar to `dump()` and we can therefore evaluate them without any problems.

C4) *Behavior*. As all functions in Iris are deterministic and we evaluate them only if their arguments are reference-free, their result is indifferent no matter if they are evaluated now or later.
The restriction to reference-free arguments is necessary as exemplified by Figure 6. If both properties in "`Block A`" are selected, we can not simply transform "`Block B.Unselected`". Because when we simplify the value to `24`, further changes of "`Block A.Selected1`" do no longer affect "`Block A.Selected2`".

T2 Inline Definitions The inline definitions transformation is executed in the *outer* and *inner* obfuscation subphases. In the outer phase, only unselected constraints are considered as targets for inlining, in the inner phase it is allowed to inline values from unselected to selected constraints (but never to inline the other way around). It is similar to the constant propagation compiler optimization (Cooper & Torczon 2011, Chp. 8). As Iris does not allow user-defined functions with its domain-specific language, we do not consider the inlining of functions (Chang & Hwu 1989).

To ensure that we can safely inline a value, we have to be sure that we lose no impact (cf. T7 Remove Orphaned). With the constraints (*i*) `B = [5..6]`, (*ii*) `B = 5`, and (*iii*) `A = B + 2` we can not inline (*i*) into (*iii*), as we would lose the impact of (*ii*) on (*iii*) (as Iris offers no mechanism to trace that the inlined interval is connected to `B`).

To solve this problem, we refer to the solver which can infer the closest bounds for each variable. If those bounds are not dependent on the currently simulated time, we can simply inline the closest bound (`B = 5` in the example above). Otherwise, we have to perform an impact analysis—which uses the data flow of the expression language—to retrieve all constraints that affect `B` in `A = B + 2` and only inline if there is no other constraint impacting the reference to inline.

With this, we can define the transformation as follows: For each selected and unselected reference r (e.g. a reference to a property) in each unselected constraint c in the *outer* subphase and each unselected reference r in each selected constraint c in the *inner* subphase: If there is a value v for r that can be inferred without removing any necessary impact, replace r in c with v.

We use a *deterministic* variant of the inline definitions transformation by inlining whenever possible. If one restricts the approach to only apply inline definitions, the transformation may not be necessary to be *reversed* by an attacker. That is, while it conceals information about the origin of values, it does not remove the formulas the inlined value is based on. As an example, consider a property `Factor` which is assigned the value `3` and intellectual property that is to be removed. Simply inlining `Factor` into the constraint `Outcome = Factor * Input` to `Outcome = 3 * Input` does not remove the value `3` from the model. However, by inlining all the direct effects of such a formula, this transformation can make them obsolete for the behavior so they get removed by T7 Remove Orphaned. Furthermore, the inlined values can be diluted further by applying T1 Arithmetic Simplification.

It may be required to check for the effects of the currently simulated time to determine the impact. As this transformation has to look at other constraints to extract values, it is *global*.

We argue for the following effects on the four constraints by applying inline definitions:

C1) *Self-Containment*. Because this transformation only removes references, it does not reintroduce any reference to $a \in \mathbb{E}_A$. However, this behavior may change if we allow the expressions to be inlined to contain additional references. Nevertheless, all of them can be obfuscated by T6 Identifier Obfuscation leaving no reference to A.

C2) *Selected*. The transformation does not use selected variables as a basis for inlining. Therefore, it only makes unselected constraints applicable for removal by T7 Remove Orphaned, never removing selected elements.

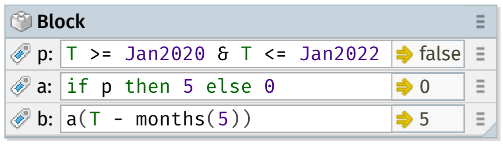
C3) *Inference*. This transformation removes origin information but retains the information of unselected elements. However, as discussed alongside the reversibility, the application of this transformation lays the groundwork for removing inferable information by other transformations.

C4) *Behavior*. Because we only inline values whenever it does not break the impact of other constraints, applying inline definitions does not affect behavior.

T3 Expression Normalization The expression normalization transformation is required to reorder constraints into a standardized form to drastically reduce the complexity of the patterns to be matched by other transformations and to remove information potentially embedded in the ordering of operations. We use this transformation to *deterministically* normalize constraints to an Iris-internal normal form (Breckel et al. 2021).

For each selected and unselected constraint, this transformation exchanges the order (and sometimes the type of) operations while preserving behavior (e.g., `2 < x + 2` to `2 - 2 < x`).

Because we only change the order of operations, there is nothing that has to be reversed, making the transformation *reversible*. Nevertheless, by transforming all constraints into a standardized form, we can still hide some intent embedded in the original notation. This transformation neither changes model elements nor does it affect behavior. Therefore, we do not discuss the constraints in any greater detail, as C1–C4 remain unaffected.



Block

```
p: T >= Jan2020 & T <= Jan2022   ⇒ false
a: if p then 5 else 0            ⇒ 0
b: a(T - months(5))              ⇒ 5
```

Figure 7 Tailoring with local time dependence in IRIS. The set time is April 2022. Suppose we restrict the domain of global T to lie between January 2021 and December 2023. In that case, we can not simply tailor the expression as described, as b accesses a for a different local time.

5.2.2. Disguising the Result Transformations of this group hide details about the results themselves by exploiting several domain limitations of external or internal factors (i.e., variables). This applies for infinite domains (e.g., $[3, \infty)$) as well.

T4 Tailor Expressions The goal of the tailor expression transformation is to tailor functions to the domains of their input vectors. Consider an expression like `linear(x, 0, 5, 10, 100)` which interpolates a result for the value of x between 5 for $x \le 0$ and 100 for $x \ge 10$. For $x = 8$ this expression evaluates to $5 + (100 - 5)/10 \cdot 8 = 81$. If x is limited to $x \in [0, 8]$, we can tailor the given linear expression to `linear(x, 0, 5, 8, 81)`, hiding calculation details outside of the domain and therefore impeding extrapolation.

Considering time in IRIS, there are two features affecting this transformation: (i) the possible values for the global time can be restricted by a closed interval, and (ii) the global value of T can be shadowed when using references. The restriction of the global time can be seen to the left and right of the time slider in Figure 1—dates before 2019 and after 2040 (which is the exclusive upper limit) are not considered for this model. Shadowing by using a different local time allows referencing "The Car.Demand" from Figure 1 as `The Car.Demand(T - months(2))` (⇒ 4.4 for the currently simulated time of Dec 2022). This does not impede tailoring in general—it even benefits the process—however, it requires greater care when tailoring an expression.

Consider the example in Figure 7. The global time T is restricted to lie between January 2021 and December 2023. If we just look at the property "p", tailor expression could try to replace Jan2020 by Jan2021 (as dates before January 2021 are not considered due to the time restriction). However, this would result in a wrong evaluation for b which uses `a(T - months(5))` to access a in the time interval `[Aug2020..Jul2023]`.

Hence, we can not tailor the expression without taking into account all potential local bindings for T. We address this issue by using the solver to infer the closest domain that contains all possible values for the local binding of T. In this example, the closest domain would be the union of both intervals: `[Aug2020..Dec2023]`.

For tailor expressions to work, we provide a predefined tailoring function for each IRIS-function that is applicable for tailoring. All of these tailoring functions ensure termination. For example, they take care to prevent replacing the expression e with $f(e)$ (which may result in an infinite recursion if $f(e)$ is to be tailored again). The tailor functions can access the closest domain for all references and external factors as inferred by the solver. As these functions are *deterministic* and because there is only one tailoring function per applicable IRIS-function, the property propagates to the tailor expressions transformation.

In general, this transformation is *irreversible* albeit external semantics by the modeling domain may allow educated guesses. If we know that the function is linear for $[a, b]$ and $[c, d]$ with $b < c$ it may be safe to guess the values for $[b, c]$ and use this information to extrapolate for values outside of $[a, d]$. Because we need the closest domain for all references in question and for T, the transformation is considered to be *global*.

We argue for the following effects on the four constraints:

C1) *Self-Containment*. This transformation updates only parts of an expression, with the tailoring functions never creating any unknown reference. Therefore, tailor expressions is incapable of invalidating this constraint. Otherwise, it has the same problems as T1 Arithmetic Simplification (regarding the addition of new functions to IRIS).

C2) *Selected*. Because this transformation does not perform any structural modification, it can not alter the elements contained in the output model.

C3) *Inference*. As we have already argued with the reversibility of this transformation, its application does impede potential extra- and interpolation.

C4) *Behavior*. The tailor expression transformation only removes information for inputs that lie outside of the domain of the global time. Because behavioral equivalence respects this domain, removing information outside of it does not affect it.

T5 Reduce Constants The goal of the reduce constant transformation extends on T2 Inline Definitions. While the latter one restricts itself to inline only if we know that we do not lose any calculation information, reducing constant makes use of the fact that we know constraints outside of the selection are not subject to change.

For example, consider two constraints in the form `x >= 2` and `x <= 5` that require x to lie in the closed interval `[2..5]`. If both constraints are unselected and therefore not be changed, we can simply intersect the domain of x by their restrictions. If x is currently $(-\infty, \infty)$ (i.e., unbound), this would result in a new constraint: `x = [2..5]`. In other words: we replace the value for x with the closest domain inferred by the solver (as long as the global time is not affecting this closest domain, cf. T2 Inline Definitions). After the application of the reduce constants transformation, the value x is applicable for inlining by T2 Inline Definitions.

We can describe the transformation like this: For every unselected (outer subphase) and selected reference (inner subphase) r in the constraints, infer the closest possible bound β_r using only unselected constraints. For each β_r, produce a new unselected constraint $r = (\beta_r)$. Because there is only one closest possible β_r for each reference, we consider the transformation to be *deterministic*. By considering all unselected constraints to infer the closest bound, this transformation is *global*.

Due to the transformation not modifying the already existing constraints in the model we can summarize the effects on the four constraints based on the arguments before, as none of the constraints are contradicted.

5.2.3. Disguising Structure

Transformations of this group deal with the structural modification of a model, including the obfuscation of reference, hierarchy, and usage information. The most important of those transformations is T7 Remove Orphaned, which removes constraints and model elements that are no longer required. Additionally, this phase is responsible for the recreation of a model based on the constraints.

T6 Identifier Obfuscation The goal of the identifier obfuscation transformation is the exchange of identifiers (i.e., the names of properties, blocks, ...) with new and randomly created ones so that it is impossible to reason about the original identifiers.

For this transformation to work, we add a mapping function m in the preparation phase (cf. Section 5), that maps identifiers from the input model to the obfuscated output.

We can describe the transformation like this: For each reference r in the constraint, we check first if the mapping function m does already contain a mapping for r. If it does, we replace r by $m(r)$, otherwise, we randomly generate a new *unique* identifier r', store it in m so that $m(r) = r'$ and, replace r with r'.

Because there is no way of linking the newly generated identifiers to the input model, this transformation is *irreversible*.

It should be noted that it is possible to create a deterministic variant of the identifier obfuscation. For example, by incrementing a counter for each newly encountered reference (we use this within the examples to keep them consistent). As long as the process itself does not allow an attacker to make any guesses about, e.g., the number of elements in the original model, the impact of this transformation remains the same—in IRIS we have implemented both variants.

Applying the identifier obfuscation on the four constraints has the following consequences:

C1) *Self-Containment*. This transformation creates only *new* references that, by construction, are unable to reference any $a \in \mathbb{E}_A$.
C2) *Selected*. This transformation does neither remove nor add a new model element. Yet, it exchanges their unique identifiers.
C3) *Inference*. By removing potential information introduced by aliases or the identifier generation process of the modeling language, this transformation reduces the potentially inferable information.
C4) *Behavior*. As this transformation exchanges all identifiers consistently, this does not affect the behavioral equivalence.

T7 Remove Orphaned This remove orphaned transformation is critical for hiding unwanted information as it is responsible for the removal of constraints that have no longer any influence on the behavior of the selected variable. It uses impact analysis (similar to T2 Inline Definitions) to identify constraints that do not affect any selected constraint.

This method *may determine*, depending on the method we use to identify the impact of unselected constraints. In general, it does not have to. Whenever there are two redundant constraints like `x >= 3` and `x >= 3` whereas either one of them affects the domain of `x`, while the other one does not—the one to remove can be chosen randomly. Because we remove the constraints completely, we consider the transformation to be *irreversible*.

Removing orphans has the following consequences on the four constraints:

C1) *Self-Containment*. Because we only remove constraints we reduce the number of references to $a \in \mathbb{E}_A$ at most.
C2) *Selected*. As only unselected constraints are removed, nothing selected is affected.
C3) *Inference*. In combination with other transformations, like T2 Inline Definitions, this transformation is key in impeding the inferable information on unselected elements.
C4) *Behavior*. By construction, we only perform removals that do not affect the behavioral equivalence.

T8 Rebuild Hierarchy The goal of the rebuild hierarchy transformation is to create the output model based on the set of constraints transformed by the previously employed transformations. While its main purpose is to produce an IRIS-model, it serves the purpose of creating a new hierarchy, ensuring that the constraints C1–C4 hold.

When recreating the model hierarchy there are several special cases to deal with. Take a look at Figure 1 and assume that the blocks "The Car" and "Wheel" are selected. When recreating the model, the unselected "Chassis" block—the parent block of "Wheel" and the child block of "The Car"—should not be recreated. When removing "Chassis" we must re-parent "Wheel" *and* deal with all references like `Chassis.Wheel.Diamater` in the "Top Speed"-property of "The Car".

As another example, consider constraints that can not be fully removed to keep the behavior of the output model intact. We can not simply add them to the model because, in IRIS, all properties, requirements, and constraints need a block that contains them (cf. Figure 2). Therefore, we produce a new so-called "Collector" block to contain all these constraints (cf. Figures 3 and 5).

As a third case, note that IRIS does not create constraints for elements that are neither referenced nor assigned to an expression which would invalidate C2. To cope with this, we manually ensure the existence of all selected model elements by an extra pass over the selection, recreating all missing elements. In Figure 5, this results in the creation of the property "Collector.P4" so that the formula of "Block1.A" can reference it.

Furthermore, there is an essential point to be made about implicit assumptions, as they can make the process more difficult if they can not be expressed directly in the modeling environment. Consider the Availability-property in IRIS. As it is generated based on the hierarchy of the model (including the requirements of child blocks), we have a problem if some of these requirements are not selected as we have to remove them due to C2 (*Selected*) but keep their influence to satisfy the fourth constraint (C4, *Behavior*). To deal with this problem, we have extended the IRIS expression language so that it is capable of expressing all implicit assumptions (like the Availability-property) explicitly by special model elements. This allows us to remove unselected requirements and blocks while retaining their impact within these special elements, satisfying both C2 and C4.

Because this transformation finally creates the output model in question, the discussion about its effects on the four constraints is of special interest:

C1) *Self-Containment.* Assuming, T6 Identifier Obfuscation already operated on the constraints (which it does with our execution order as explained in Section 5.3), all unique identifiers are newly generated. Therefore, the model does not reference or use any model element from A.

C2) *Selected.* The effect on the selected elements depends on the transformations employed before. If all of them remain in the model—which is what we expect—this transformation ensures that all $e \in S$ are present.

C3) *Inference.* The major purpose of this transformation is to produce a model and not to reduce the inferable information. Therefore, it does not remove any information. On the contrary, it may even re-induce some information by creating hierarchical structures removed by the selection. Nonetheless, if the exact model structure is crucial for the modeling language syntax, this can not be helped.

C4) *Behavior.* By essentially just reformatting the constraints to be set inside an IRIS-model that re-generates similar constraints, we do not change the behavioral equivalence. All implicit assumptions—like the Availability-property of blocks—are now expressed explicitly whenever necessary.

T9 Remove Implicit The remove implicit transformation is of use if any implicit assumptions—like the Availability property of blocks—remain in the model produced by T8 Rebuild Hierarchy. In other words, it removes each (unselected) model element that is already covered by implicit assumptions. For the availability properties, we simply compare the normalized constraint with the implicitly created one.

Similar to T7 Remove Orphaned, there may be situations in which, for example, one of two redundant model elements may be chosen at random. Hence, the transformation may be *indeterministic*. However, we only remove model elements that are already covered by implicit assumptions.

As we do not remove any information from the model but rather hide information that could be used to identify what has been obfuscated, we consider this transformation to be *reversible*.

The transformation is *global* because we have to consider the impact of other model elements in combination with their hierarchy to determine if they are covered by implicit assumptions. Removing the implicit assumptions has the following impact on the four constraints:

C1) *Self-Containment.* Because the model is already built by T8 Rebuild Hierarchy and because we only remove elements, we can not re-introduce references to $a \in \mathbb{E}_A$.

C2) *Selected.* We do only remove elements stemming from unselected constraints. Therefore, we never remove the elements from the selection.

C3) *Inference.* Using the same arguments made for the transformations' reversibility, we do neither improve nor impede the theoretically inferable information on any $e \in \mathbb{E}_A \setminus S$.

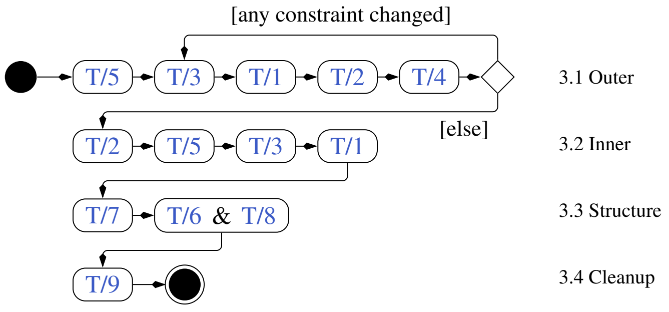


Figure 8 Activity diagram of a potential execution order of the transformations. Each line represents one of the obfuscation subphases 3.1–3.4 (as indicated by the small numbers to the right).

C4) *Behavior.* By only removing model elements that are covered by implicit assumptions we do not affect the behavior.

5.3. Execution Order of Transformations

While there are many possible execution orders for the transformations presented in Section 5.2, Figure 8 gives a brief overview of the order implemented in IRIS. Starting with T5 Reduce Constants, setting (whenever possible) the closest bounds for all constants. After that, we repeatedly normalize the constraints (T/3), simplify and inline where applicable (T/1 & T/2) followed by tailoring them (T/4), until we reach a fixpoint (i.e. the constraints do not change anymore). Moving on to the inner obfuscation subphase, we start with T2 Inline Definitions, inlining values of unselected model elements into selected elements whenever applicable, calculate the closest bound for all selected references by using only unselected constraints (T/5), normalize the constraints (T/3), and finally simplify inlined parts in the selected constraints (T/1). For the structure obfuscation subphase, we first remove all orphaned constraints (T/7), followed by a combined application of T8 Rebuild Hierarchy and T6 Identifier Obfuscation (i.e., T/6 is executed as a part of T/8). After that, we run T9 Remove Implicit on the newly created model.

5.4. Obfuscation Workflow

Within IRIS, we decided to restrict the input selection to blocks—automatically selecting all of the properties, requirements, and constraints they contain—to improve usability.

To exchange relevant information with the fuel tank manufacturer, the OEM of the running example in Section 2 would select the "Fuel Tank"-block and start the obfuscation process as explained at the start of Section 5. This results in a preview of the obfuscated model as shown in Figure 3. In this preview, the OEM can review the export and manually validate whether only relevant information and no intellectual property are included. If desired, the OEM can change the exported model, supported by an automatic and continuous evaluation of C1, C2, and C4. Furthermore, the OEM can export and send the obfuscated model to the fuel tank manufacturer which in turn imports the obfuscated model into its model. Now, the fuel

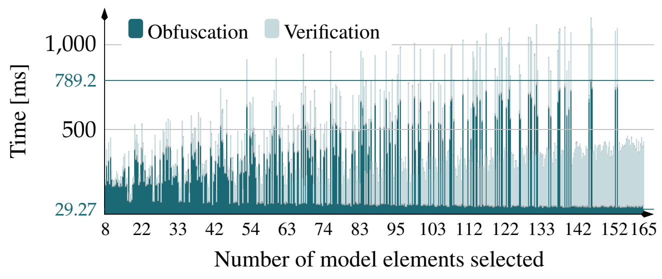


Figure 9 Required time to perform the obfuscation (phases 1–3) and the verification (phase 4) with a variable number of model elements selected.

tank manufacturer models one or many solutions which exactly provide the information requested by the OEM and also can aim at fulfilling the given requirements and constraints. After this, the fuel tank manufacturer runs the obfuscation process to not share IP with the OEM and sends back the exported fuel tanks to the OEM. The OEM, finally, imports the provided fuel tanks into the OEM model and evaluates the provided solution (in place).

6. Evaluation

Our main goal was to answer RQ1 to perform a technical feasibility study of the respective transformations. For RQ1, we already analyzed all transformations regarding the constraints C1–C4 and discussed their properties from Section 5.1 as summarized in Table 1.

For the implementation, we created an automated check which validates the behavioral equivalence of two models to verify C4 explicitly (cf. the verification phase in Section 5). We use this check to validate every possible selection for a given set of models—consisting of IRIS-examples and specifically crafted edge cases—as well as randomly generated models. Four of the large IRIS-examples in use have been crafted by domain experts of our industry partners. By passing these tests, we gain a baseline confidence for the effectiveness of our approach regarding C1, C2, and C4.

Furthermore, we created 204 unit tests which achieve a (statement) test coverage of 94.7 %, systematically covering the obfuscation of all language features of IRIS—including their edge-cases—in combination with every possible input selection to ensure the quality of our implementation. Unchecked statements contribute to type-safety and the IRIS-frontend.

To assess if the implementation in IRIS is applicable, we measured the performance by benchmarking selections of various sizes for different models using an average office computer. The benchmarking revealed maximum obfuscation times ranging between 16.32 ms for small models of 32 model elements and 789.2 ms for a larger model with 165 model elements—the full plot of the benchmark for the latter model can be examined in Figure 9. It shows greatly varying obfuscation times. This variation is attributed to the (naïvely implemented) impact analysis required by T7 Remove Orphaned and T2 Inline Definitions (therefore, it depends on the number of unselected time-dependent constraints that require this analysis as explained with T/2). As expected, the verification time scales linearly with the number of elements both models have in common.

Regarding the overall effectiveness, we conducted several demonstrations with domain experts from the industry using example scenarios, with subsequent discussions and feedback rounds. We can report positive feedback from our industry partners. However, a case study regarding C3 with real-world scenarios (e.g., domain experts trying to reverse engineer the original model from the obfuscated output) remains future work.

7. Discussion

In examining the approach's applicability, we will first investigate its limitations in practice in Section 7.1, followed by a discussion of its generalizability in Section 7.2.

7.1. Limits

Furthermore, we used the publications gathered during our research (cf. Section 4) as a basis for a discussion of the theoretical limitations of the proposed approach. Even considering all nine transformations presented, they can not obfuscate every model for every selection to not reveal any more than oracle access for unselected elements—the existence of a general obfuscator achieving this has been proven to be impossible by Barak et al. (2001). We have examined the following four limitations to be the most serious:

1. We can not prevent attackers from having a more detailed understanding of the domain. As an example, consider a domain expert in the automotive industry. Even though the concrete formulas for the wheels of the cars may be obfuscated, their indirect effect on the behavior of the remaining model may be enough to identify known key characteristics of a specific wheel manufacturer.
   Additionally, competition laws could pose a problem as well. Even law experts have difficulties evaluating specific scenarios because "the legality of a certain conduct is often also dependent on factual circumstances which might be difficult to evaluate" (BusinessEurope 2017, p. 8)—so whether a specific obfuscation is "compliant" may be decided by legal decisions on a case-by-case basis.
2. For some functions (like point functions), it has been proven that it is impossible to obfuscate them into a form that reveals nothing more than oracle access (Narayanan & Shmatikov 2006; Wee 2005; Barak et al. 2001).
3. While we have based our properties on other work focused on characterizing obfuscating transformations and model transformations in general (Amrani, Dingel, et al. 2012; Amrani, Lucio, et al. 2012; Collberg et al. 1997), they could be insufficient in identifying all relevant aspects.
4. Furthermore, we did not focus on transformations that deal with the obfuscation of data. This could result in further problems when applying additional transformations dealing with that (cf. Section 8).

7.2. Generalizability

While we introduce all transformations in the context of IRIS, a lot of them are useful in the context of other tools as well.

Nevertheless, future research is necessary regarding their application. The most obvious alternative is Microsoft Excel (as well as other spreadsheet tools like LibreOffice Calc), as IRIS shares a lot of similarities regarding the expression language and the semantics. Moreover, as indicated by or industry partners, Microsoft Excel is a prominent tool in the context of technology roadmapping (Rinne 2004). By interpreting the input as well as the output model as a complete spreadsheet, and a selection of elements as a selection of cells (identified by their position in the spreadsheet or an accompanying cell), T/1 – T/3 , and T/5 – T/7 can be adapted directly. Two things to care about are:

- Functions like `RAND()` which produce a different result with each calculation. If they are to be evaluated at obfuscation time may be dependent on the context.
- Functions like `WEBSERVICE(url)`, custom functions in other languages, or anything that deals with information from the outside world may introduce unforeseen consequences that have to be dealt with separately (as we did not have to deal with these problems in the context of IRIS).

However, since Microsoft Excel does not have intervals as they are present in IRIS, T4 Tailor Expressions can not be directly adapted in the context of Excel formulas as it uses the concept of intervals, although it could be used in combination with data validation that restricts the values of cells. Similarly, T8 Rebuild Hierarchy could be adapted for cells that are used in combination. For example, one cell describes a name and the other cell describes the value. Regarding the obfuscation of data, this could be combined with AnonymousXL by Van Veen & Hermans (2014).

Another more different use case would be, for example, the obfuscation of Modelica models (Otter 2000), which allow the calculation of values, connecting them with connectors. Interpreting the selection as a selection of basic models, T/1 – T/3 , and T/5 – T/7 can be applied to the equation sets within these models, similar to the constraints in IRIS, with the connectors linking variables. T/8 can be adapted to reconstruct the hierarchy of the obfuscated output model, and T/9 to remove asserts that are covered implicitly.

8. Conclusion and Future Work

In this paper, we addressed the problem of removing content worth protecting from a model while leaving the behavior of selected elements intact, by proposing a four-phase transformation program that applies nine one-way model transformations in the context of the graphical roadmapping tool IRIS. For this, we derived four constraints so that the output model is self-contained (C1), contains the selected elements (C2), allows no inference on the unselected elements (C3), and is behavioral equivalent to the input model. For RQ1 we selected nine relevant transformations (cf. Table 1) and evaluated them according to their properties and their effects on C1–C4.

Furthermore, we evaluated our implementation in Section 6, demonstrating that the approach is applicable in practice. While we were able to show that these transformations are capable of satisfying C1, C2, and C4, they are insufficient of additionally satisfying C3 completely (cf. Section 7.1). To address C3 we plan on extending the approach by several other transformations that (Sihler 2023):

- *generate noise*, like other model elements and expressions that do not change the selected behavior.
- *approximate functions* by other functions, allowing to obfuscate unselected formulas (that are critical for the behavioral equivalence) by approximating replacements.
- *distort individual data* as done by Bakken et al. (2004), Van Veen & Hermans (2014), and others.
- *modify structure* by merging and splitting blocks (e.g., the collector block).

Furthermore, we plan on evaluating the effectiveness of the existing and new transformations by examining reverse-engineering attacks by domain experts.

Acknowledgments

We would like to thank Alexander Breckel for countless fruitful discussions and valuable inputs, especially regarding the internals of the IRIS-solver.

This work has been developed in the project *GENIAL!* (reference number: 16ES0875). GENIAL! is partly funded by the German Federal Ministry of Education and Research (BMBF) within the research programme ICT 2020.

In this article, figures show screenshots of IRIS. The screenshots include icons of the following authors: some icons by Yusuke Kamiyamane, https://p.yusukekamiyamane.com/, licensed under a Creative Commons Attribution 3.0 License, (https://creativecommons.org/licenses/by/3.0/). Further, some icons are taken from FatCow, https://www.fatcow.com/free-icons, licensed under a Creative Commons Attribution 3.0 United States License, https://creativecommons.org/licenses/by/3.0/us/.

References

Ab. Rahim, L., & Whittle, J. (2015, May). A survey of approaches for verifying model transformations. *SoSyM' 15*, *14*(2), 1003–1028. doi: 10.1007/s10270-013-0358-0

Amrani, M., Dingel, J., Lambers, L., Lúcio, L., Salay, R., Selim, G., ... Wimmer, M. (2012). Towards a model transformation intent catalog. In *AMT' 12* (pp. 3–8). New York, NY, USA: ACM. doi: 10.1145/2432497.2432499

Amrani, M., Lucio, L., Selim, G., Combemale, B., Dingel, J., Vangheluwe, H., ... Cordy, J. R. (2012). A tridimensional approach for studying the formal verification of model transformations. In *ICST '12* (p. 921-928). doi: 10.1109/ICST.2012.197

Bakken, D. E., Rarameswaran, R., Blough, D. M., Franz, A. A., & Palmer, T. J. (2004). Data obfuscation: Anonymity and desensitization of usable data sets. *IEEE Security & Privacy*, *2*(6), 34–41. doi: 10.1109/MSP.2004.97

Balakrishnan, A., & Schulze, C. (2005). Code obfuscation literature survey. *CS701 Construction of compilers*, *19*.

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2001). On the (im)possibility of obfuscating programs. In *CRYPTO 2001* (pp. 1–18). Springer Berlin Heidelberg.

Bergmann, G., Debreceni, C., Ráth, I., & Varró, D. (2016). Query-based access control for secure collaborative modeling using bidirectional transformations. In (pp. 351–361). New York, NY, USA: ACM. doi: 10.1145/2976767.2976793

Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., … others (2012). Functional mockup interface 2.0: The standard for tool independent exchange of simulation models..

Breckel, A., Pietron, J., Juhnke, K., Sihler, F., & Tichy, M. (2021). A domain-specific language for modeling and analyzing solution spaces for technology roadmapping. *JSS*, *184*, 111094. doi: 10.1016/j.jss.2021.111094

BusinessEurope. (2017, April). Making sense of competition law compliance – a practical guide for SMEs [Computer software manual].

Chang, P. P., & Hwu, W.-W. (1989, June). Inline function expansion for compiling C programs. *SIGPLAN Not.*, *24*(7), 246–257. doi: 10.1145/74818.74840

Collberg, C., Thomborson, C., & Low, D. (1997). *A taxonomy of obfuscating transformations* (Tech. Rep.). Citeseer.

Cooper, K., & Torczon, L. (2011). *Engineering a compiler*. Elsevier.

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM systems journal*, *45*(3), 621–645.

Debreceni, C. (2019). *Advanced techniques and tools for secure collaborative modeling* (phdthesis). Budapest University of Technology and Economics.

Debreceni, C., Bergmann, G., Búr, M., Ráth, I., & Varró, D. (2017). The MONDO collaboration framework: Secure collaborative modeling over existing version control systems. In (pp. 984–988). New York, NY, USA: ACM. doi: 10.1145/3106237.3122829

Debreceni, C., Bergmann, G., Ráth, I., & Varró, D. (2018). Secure views for collaborative modeling. *IEEE Software*, *35*(6), 32–38. doi: 10.1109/MS.2018.290101728

Fill, H.-G. (2012). Using obfuscating transformations for supporting the sharing and analysis of conceptual models. Braunschweig: GITO Verlag.

Garcia, M. L. (1997). *Introduction to technology roadmapping: The semiconductor industry association's technology roadmapping process*. Sandia National Laboratories.

Garcia, M. L., & Bray, O. H. (1997). *Fundamentals of technology roadmapping* (Tech. Rep.). Sandia National Labs., Albuquerque, NM (United States).

Goettelmann, E., Ahmed-Nacer, A., Youcef, S., & Godart, C. (2015). Paving the way towards semi-automatic design-time business process model obfuscation. In *IEEE ICWS '15* (p. 559-566). doi: 10.1109/ICWS.2015.80

Gupta, N., Chen, F., Tsoutsos, N. G., & Maniatakos, M. (2017). Obfuscade: Obfuscating additive manufacturing cad models against counterfeiting: Invited. In *DAC '17.* New York, NY, USA: ACM. doi: 10.1145/3061639.3079847

Hosseinzadeh, S., Rauti, S., Laurén, S., Mäkelä, J.-M., Holvitie, J., Hyrynsalmi, S., & Leppänen, V. (2018). Diversification and obfuscation techniques for software security: A systematic literature review. *Information and Software Technology*, *104*, 72-93. doi: 10.1016/j.infsof.2018.07.007

Karagiannis, D., & Kühn, H. (2002). Metamodelling platforms. In (Vol. 2455, p. 182).

Narayanan, A., & Shmatikov, V. (2006). On the limits of point function obfuscation. *IACR Cryptol. ePrint Arch.*, 182.

Otter, M. (2000). Modelica-a unified object-oriented language for physical systems modeling-language specification.

Phaal, R., Farrukh, C. J., & Probert, D. R. (2001). Characterisation of technology roadmaps: purpose and format. In *PICMET '01* (Vol. 2, pp. 367–374). doi: 10.1109/PICMET.2001.952036

Pietron, J., Funk, L., & Tichy, M. (2022). Improving the comprehension of evolving graphical models. In *VISSOFT '22* (pp. 96–107). IEEE. doi: 10.1109/VISSOFT55257.2022.00018

Prasser, F., Kohlmayer, F., Lautenschläger, R., & Kuhn, K. A. (2014). ARX — a comprehensive tool for anonymizing biomedical data. In *AMIA annual symposium proceedings* (Vol. 2014, p. 984).

Preuß, M. (2016). *Inference-proof materialized views* (Unpublished doctoral dissertation).

Rinne, M. (2004). Technology roadmaps: Infrastructure for innovation. *Technological Forecasting and Social Change*, *71*(1), 67-80. doi: 10.1016/j.techfore.2003.10.002

Sihler, F. (2023). *One-way model transformation: On structural and semantical one-way transformations for constraint-based models, exemplified in the context of roadmapping* (Bachelor's Thesis). doi: 10.18725/OPARU-47275

Sosonkin, M., Naumovich, G., & Memon, N. (2003). Obfuscation of design intent in object-oriented applications. In *ACM DRM '02* (pp. 142–153). New York, NY, USA: ACM. doi: 10.1145/947380.947399

Van Veen, J., & Hermans, F. (2014). Anonymizing spreadsheet data and metadata with anonymousxl. *SEMS '14*, 50–51.

Wee, H. (2005). On obfuscating point functions. In *STOC '05* (pp. 523–532).

Yücesan, E., & Schruben, L. (1992). Structural and behavioral equivalence of simulation models. *TOMACS*, *2*(1), 82–103.

Zhou, B., Pei, J., & Luk, W. (2008, dec). A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, *10*(2), 12–22. doi: 10.1145/1540276.1540279

About the authors

Florian Sihler is a student at Ulm University (Germany), currently working on his master's degree. You can contact the author at florian.sihler@uni-ulm.de.

Matthias Tichy is full professor for software engineering at Ulm University and director of the Institute of Software Engineering and Programming languages. His main research focuses on model-driven software engineering, particularly for cyber-physical systems. You can contact the author at matthias.tichy@uni-ulm.de.

Jakob Pietron is a Ph.D. candidate at the Institute of Software Engineering and Programming Languages, Ulm University, Germany. You can contact the author at jakob.pietron@uni-ulm.de.