A Categorical Semantics for Hierarchical Petri Nets

Fabrizio Genovese
0000-0001-7792-1375
University of Pisa
Statebox
fabrizio.romano.genovese@gmail.com

Jelle Herold
Statebox
research@statebox.io

Fosco Loregian
0000-0003-3052-465X
Tallinn University of Technology
fosco.loregian@gmail.com

Daniele Palombi
0000-0002-8107-5439
Sapienza University of Rome
danielepalombi@protonmail.com

We show how a particular flavor of hierarchical nets, where the firing of a transition in the parent net must correspond to an execution in some child net, can be modelled utilizing a functorial semantics from a free category – representing the parent net – to the category of sets and spans between them. This semantics can be internalized via Grothendieck construction, resulting in the category of executions of a Petri net representing the semantics of the overall hierarchical net. We conclude the paper by giving an engineering-oriented overview of how our model of hiearchic nets can be implemented in a transaction-based smart contract environment.

1 Introduction

This paper is the fourth installment in a series of works [7, 6, 5] devoted to describing the semantics of extensions of Petri nets using categorical tools.

Category theory has been applied to Petri nets starting in the nineties [14]. The main idea is that we can use different flavors of free monoidal categories to describe the executions (or runs) of a net [13, 8]. The reason why this has been influential is that it opened up a avenue of applying high-level methods to the study of Petri nets and their properties. For instance, in [1] the categorical approach allowed to describe gluing of nets leveraging on colimits and double categories, while category-theory libraries such as [9] can be leveraged to implement nets in a formally verified way.

In [7], we started another line of research, where we were able to define a categorical semantics for colored nets by means of monoidal functors. The Grothendieck construction was then used to internalize this semantics, obtaining the well-known result that colored nets can be "compiled back" to Petri nets.

In [6, 5], we extended these ideas further, and we were able to characterize bounded nets and mana-nets – a new kind of nets useful to model chemical reactions – in terms of generalized functorial semantics.

This approach, based on the correspondence between slice categories and lax monoidal functors to the category of spans [15], has still a lot to give. In this paper, we show how it can be used to model hierarchical nets.

There are a lot of different ways to define hierarchical nets. For us, it means that we have one "parent" Petri net, and a bunch of "child" nets. A transition firing in the parent net corresponds to some kind of run happening in a corresponding child net. As such, the main net serves the purpose of orchestrating and coordinating the executions of many underlated child nets.

Submitted to:
CGM 2021

This paper will contain very little new mathematics. Instead, we will just reinterpret results obtained in [7] to show how they can be used to model hierarchical nets, moreover in a way that makes sense from an implementation perspective.

2 Nets and their executions

We start by recalling some basic facts about Petri nets and their categorical formalization.

Notation 1. *Let S be a set; a multiset is a function $S \to \mathbb{N}$. Denote with $S^{\oplus}$ the set of multisets over S. Multiset sum and difference (only partially defined) are defined pointwise, and will be denoted with $\oplus$ and $\ominus$, respectively. The set $S^{\oplus}$ together with $\oplus$ and the empty multiset is isomorphic to the free commutative monoid on S.*

Definition 1 (Petri net)**.** *A* Petri net *is a pair of functions $T \xrightarrow{s,t} S^{\oplus}$ for some sets T and S, called the set of places and transitions of the net, respectively. s,t are called* input and output *functions, respectively, or equivalently* source and target.

A morphism of nets *is a pair of functions $f : T \to T'$ and $g : S \to S'$ such that the following square commutes, with $g^{\oplus} : S^{\oplus} \to S'^{\oplus}$ the obvious lifting of g to multisets:*

$$
\begin{array}{ccccc}
S^{\oplus} & \xleftarrow{s} & T & \xrightarrow{t} & S^{\oplus} \\
{\scriptstyle g^{\oplus}}\downarrow & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle g^{\oplus}} \\
S'^{\oplus} & \xleftarrow[s']{} & T' & \xrightarrow[t']{} & S'^{\oplus}
\end{array}
$$

Petri nets and their morphisms form a category, denoted **Petri**. *Details can be found in [14].*

Definition 2 (Markings and firings)**.** *A* marking *for a net $T \xrightarrow{s,t} S^{\oplus}$ is an element of $S^{\oplus}$, representing a distribution of tokens in the net places. A transition u is* enabled *in a marking M if $M \ominus s(u)$ is defined. An enabled transition can* fire*, moving tokens in the net. Firing is considered an atomic event, and the marking resulting from firing u in M is $M \ominus s(u) \oplus t(u)$. Sequences of firings are called* executions.

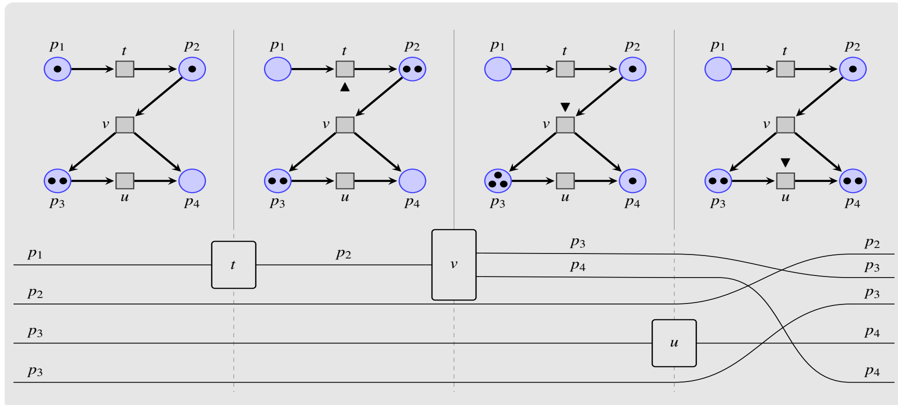
The main insight of categorical semantics for Petri nets is that the information contained in a given net is enough to genereate a free symmetric strict monoidal category representing all the possible ways to run the net. There are multiple ways to do this [16, 8, 4, 13, 2]. In this work, we embrace the *individual-token philosophy*, where tokens are considered distinct and distinguishable, and thus require the category in Definition 3 to have non-trivial symmetries.

Definition 3 (Category of executions – individual-token philosophy)**.** *Let $N : T \xrightarrow{s,t} S^{\oplus}$ be a Petri net. We can generate a* free symmetric strict monoidal category (FSSMC)*, $\mathfrak{F}(N)$, as follows:*

- *The monoid of objects is the free monoid generated by S. Monoidal product of objects A,B is denoted with $A \otimes B$.*
- *Morphisms are generated by T: each $u \in T$ corresponds to a morphism generator (u, su, tu), pictorially represented as an arrow $su \xrightarrow{u} tu$; morphisms are obtained by considering all the formal (monoidal) compositions of generators and identities.*

A detailed description of this construction can be found in [13].

In this definition, objects represent markings of a net: $A \oplus A \oplus B$ means "two tokens in A and one token in B". Morphisms represent executions of a net, mapping markings to markings. A marking is reachable from another one iff there is a morphism between them.



3 Hierarchical nets

Now we introduce the main object of study of the paper, *hierarchical nets*. As we pointed out in Section 1, there are many different ways to model hiearchy in Petri nets [10], often incompatible with each other. We approach the problem from a developer's perspective, and want to model the idea that "firing a transition" amounts to call another process, and waiting for it to finish. This is akin to calling subroutines in a piece of code. Moreover, we do not want to destroy the decidability of the reachability relation for our nets, as it happens for other hierarchical models such as the net-within-nets framework [11]. We consider this to be a very important requirement for practical reasons.

We will postpone any formal definition to Section 5. In here, we focus on giving an intuitive explanation of what our requirements are.

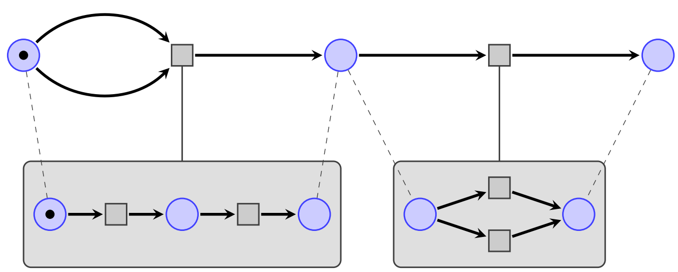


Figure 1: A hierarchical net.

Looking at the net in Fig. 1, we see a net on the top, which we call *parent*. To each transition of the parent net is attached another net, which we call *child*. We want use the parent net to orchestrate the executions of the children: Connecting input and output places of a transition in the parent net with certain places in the corresponding child, we can represent the orchestration by saying that each time a transition in the parent net fires, its input tokens are transferred to the corresponding child net, that takes them across until they reach a place connected with the output place in the parent net. This way, the atomic act of firing a transition in the parent net results into an execution of the corresponding child.

Notice that we are not interesed in considering the semantics of such hierarchical net to be akin to the one in Fig. 2, where we replaced transitions in the parent net with their corresponding children. There are two reasons for this: First, we want to consider transition firings in the parent net as atomic events, and replacing nets as above destroys this property. Secondly, such replacement is not so conceptually easy

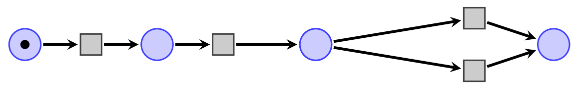


Figure 2: Replacing transitions in the parent net of Fig. 1 with its children.

given that we do not impose any relationship between the topologies of the parent net and its children. Indeed, the leftmost transition of the parent net in Fig. 1 consumes two inputs, while the corresponding leftmost transition in its child only takes one. How do we account for this in specifying a rewriting-based semantics for hierarchical nets?

4 Local semantics for Petri nets

We concluded the last section pointing out reasons that make defining a semantics for hiearchic nets less intuitive than one would initially expect. Embracing an engineering perspective, we could get away with some ad-hoc solution to conciliate the fact that parent and child net topologies are unrelated. One possible way, for instance, would be imposing constraints between shapes of the parent net and its children. But in defining things ad-hoc the possibility for unforeseen corner cases and situations we do not know how to deal with becomes high. To avoid this we embrace a categorical perspective, and define things up to some degree of canonicity.

Making good use of the categorical work already carried out on Petri nets, our goal is to leverage it and get to a plausible definition of categorical semantics for hierarchical nets. Our strategy is to consider a hierarchical net as an extension of a Petri net: The parent net will be the Petri net we extend, whereas the children nets will be encoded in the extension.

This is exactly the main tenet of [7], that is, the idea of describing net extensions with different flavors of monoidal functors. Indeed, it is our intention to show how the theory presented in [7], and originally serving a whole different purpose, can be reworked to represent hierarchical nets with minimal effort.

As semantics, we will use strict monoidal functors, and name it *local* because the strict-monoidality requirement amounts to endow tokens with properties that cannot be shared with other tokens. To understand this naming choice better it may be worth to compare it with the notion of *non-local semantics*, defined in terms of lax-monoidal-lax functors, that we gave in [6].

Definition 4 (Local semantics for Petri nets)**.** *Given a strict monoidal category $\mathcal{S}$, a* Petri net with a local $\mathcal{S}$-semantics *is a pair $(N, N^\sharp)$, consisting of a Petri net N and a strict monoidal functor*

$$N^\sharp : \mathfrak{F}(N) \to \mathcal{S}.$$

A morphism $F : (M, M^\sharp) \to (N, N^\sharp)$ is just a strict monoidal functor $F : \mathfrak{F}(M) \to \mathfrak{F}(N)$ such that $M^\sharp = F \mathbin{⨟} N^\sharp$, where we denote composition in diagrammatic order; i.e. given $f : c \to d$ and $g : d \to e$, we denote their composite by $(f \mathbin{⨟} g) : c \to e$.

Nets equipped with $\mathcal{S}$-semantics and their morphisms form a monoidal category denoted $\mathbf{Petri}^{\mathcal{S}}$*, with the monoidal structure arising from the product in* **Cat***.*

In [7], we used local semantics to describe guarded Petri nets, using **Span** as our category of choice. We briefly summarize this, as it will become useful later.

Definition 5 (The category **Span**)**.** *We denote by* **Span** *the 1-category of sets and spans, where isomorphic spans are identified. This category is symmetric monoidal. From now on, we will work with the* strictified *version of* **Span***, respectively.*

Notation 2. *Recall that a morphism $A \to B$ in* **Span** *consists of a set S and a pair of functions $A \leftarrow S \to B$. When we need to notationally extract this data from f, we write*

$$A \xleftarrow{f_1} S_f \xrightarrow{f_2} B$$

We sometimes consider the span as a function $f\colon S_f \to A \times B$, thus we may write $f(s) = (a,b)$ for $s \in S_f$ with $f_1(s) = a$ and $f_2(s) = b$.

Definition 6 (Guarded nets with side effects)**.** *A* guarded net with side effects *is an object of* $\mathbf{Petri}^{\mathbf{Span}}$. *A morphism of guarded nets with side effects is a morphism in* $\mathbf{Petri}^{\mathbf{Span}}$.

Example 1. *Let us provide some intuition behind the definition of* $\mathbf{Petri}^{\mathbf{Span}}$. *Given a net N, its places (generating objects of $\mathfrak{F}(N)$) are sent to sets. Transitions (generating morphisms of $\mathfrak{F}(N)$) are mapped to spans. Spans can be understood as* relations with witnesses*, provided by elements in the apex of the span: Each path from the span domain to its codomain is indexed by some element of the span apex, as it is shown in Fig. 3. Witnesses allow to consider different paths between the same elements. Moreover, an element in the domain can be sent to different elements in the codomain via different paths. We interpret this as* non-determinism*: The firing of the transition is not only a matter of the tokens input and output, it also includes the path chosen, which we interpret as having side-effects that are interpreted outside of our model.*

In Fig. 3 the composition of paths is the empty span: Seeing things from a reachability point of view, the

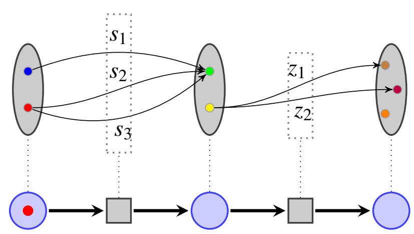


Figure 3: Semantics in **Span**

process given by firing the left transition and then the right will never occur. This can be made precise by recasting Definition 3.

Definition 7 (Markings for guarded nets)**.** *Given a guarded Petri net with side effects $(N, N^\sharp)$, a* marking *for $(N, N^\sharp)$ is a pair (X, x) where X is an object of $\mathfrak{F}(N)$ and $x \in N^\sharp X$. We say that a marking (Y, y) is* reachable *from (X, x) if there is a morphism $f : X \to Y$ in $\mathfrak{F}(N)$ such that $N^\sharp f(x) = y$.*

5 Semantics for hierarchical nets

In the span semantics we can encode externalities in the tips of the spans we send transitions to. That is, given a bunch of tokens endowed with some properties, to fire a transition we need to provide a *witness* that testifies how these properties have to be handled. The main intuition of this paper is that we can use side effects to encode the runs of some other net: To fire a transiton in the parent net, we need to provide a *trace* of the corresponding child net. This can be made precise relying on the following high-level result:

Theorem 1 ([20, Section 2.4.3]). *Given a category A with enough limits, a* category internal in A *is a monad in* $\mathbf{Span}(A)$. *Categories are monads in* **Span**, *whereas strict monoidal categories are monads in* $\mathbf{Span}(\boldsymbol{Mon})$, *with* ***Mon*** *being the category of monoids and monoid homomorphisms. A symmetric monoidal category is a* bimodule *in* $\mathbf{Span}(\boldsymbol{Mon})$, *that is, a monad in* $\mathbf{Span}(\boldsymbol{Mon})$ *with extra structure.*

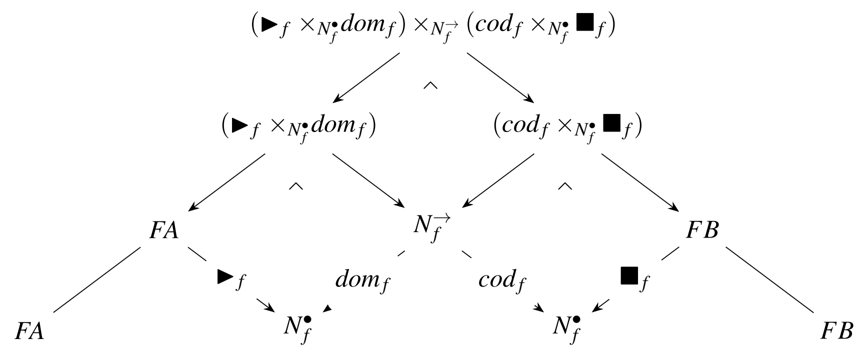
In particular, it follows that any free symmetric strict monoidal category can be represented as a span of monoids

$$N^{\bullet} \xleftarrow{\text{dom}} N^{\rightarrow} \xrightarrow{\text{cod}} N^{\bullet}$$

underlying a bimodule, with $N^{\bullet}$ and $N^{\rightarrow}$, representing the objects and arrows of the category, respectively, both free. We will refer to such a span as *the FSSMC N (in* $\mathbf{Span}(\boldsymbol{Mon})$*).*

Definition 8 (Hierarchical nets – External definition). *A* hierarchical net *is a functor* $\mathfrak{F}(N) \to \mathbf{Span}(\boldsymbol{Mon})$ *defined as follows:*

- *Each generating object A of $\mathfrak{F}(N)$ is sent to a set FA, called* the set of accepting states for the place *A.*
- *Each generating morphism $A \xrightarrow{f} B$ is sent to a span with the following shape:*



The FSSMC N_f at the center of the span is called the child net associated to f; *the morphisms $\blacktriangleright_f$ and $\blacksquare_f$ are called* play N_f *and* stop N_f, *respectively.*

Unrolling the definition, we are associating to each generating morphism of $\mathfrak{F}(N)$ – the parent net – a FSSMC – the child net. As the feet of the spans corresponding to the child nets will in general be varying with the net themselves, we need to pre- and post- compose them with other spans to ensure composability: $\blacktriangleright_f$ and $\blacksquare_f$ represent morphisms that select the *initial and accepting states* of N_f, that is, markings of N_f in which the computation starts, and markings of N_f in which the computation is considered as concluded. Notice how this also solves the problems highlighted in Section 3, as $\blacktriangleright_f$ and $\blacksquare_f$ mediate between the shape of inputs/outputs of the transition f and the shape of N_f itself.

Remark 1. *Interpreting markings as in Definition 7, We see that to fire f in the parent net we need to provide a triple (a,x,b), where:*

- *a is an element of FA, witnessing that the tokens in the domain of f provide a valid initial state for N_f.*
- *x is an element of $N_f^\rightarrow$, that is, a morphism of N_f, and hence an execution of the child net.*
- *b is an element of FB, witnessing that the resulting state of the execution x is* accepting, *and can be lifted back to tokens in the codomain of f.*

Definition 9 (Category of hierarchical Petri nets)**.** *Nets* $(N, N^\sharp)$ *in the category* $\mathbf{Petri}^{\mathbf{Span}}$ *with* $N^\sharp$ *having the shape of Definition 8 form a subcategory, denoted with* $\mathbf{Petri}^{\triangle}$*, and called* the category of hierarchical Petri nets.

Remark 2. *Using the obvious forgetful functor* $\boldsymbol{Mon} \to \mathbf{Set}$ *we obtain a functor* $\mathbf{Span}(\boldsymbol{Mon}) \to \mathbf{Span}$*, which allows to recast our non-local semantics in a more liberal setting. In particular, we could send a transition to spans whose compoents are* subsets *of the monoids heretofore considered. That is, we could select only a subset of the executions/states of the child net as valid witnesses to fire a transition in the parent.*

In doing so, everything we do in this work will go through smoothly, but we consider this approach less elegant, thus we will not mention it anymore.

6 Internalization

In Section 5 we defined hierarchical nets as nets endowed with a specific kind of functorial semantics to **Span**. As things stand now, Petri nets correspond to categories, while hierarchical nets correspond to functors. This difference makes it difficult to say what a Petri net with multiple levels of hierarchy is: Intuitively it is easy to imagine that the children of a parent net N can be themselves parents of other nets, which are thus "grandchildren" of N, and so on and so forth.

In realizing this, we are blocked by having to map N to hierarchical nets, which are functors and not categories. To make such an intuition viable, we need a way to *internalize* the semantics in Definition 8 to obtain a category representing the executions of the hierarchical net.

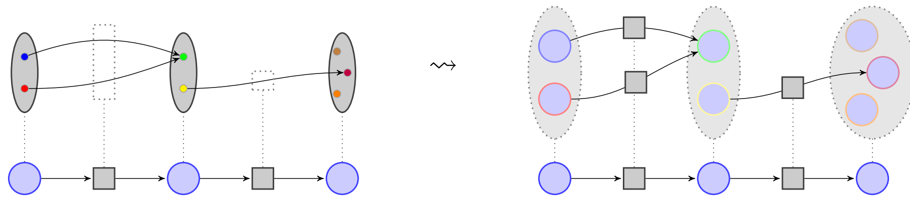
Luckily, there is a way to turn functors into categories, which relies on an equivalence between the category of slice categories over some $\mathcal{C}$ and lax-functors $\mathcal{C} \to \mathbf{Span}$ [15]. This is itself the 1-categorical version of a more general equivalence between the 2-category of slice categories over $\mathcal{C}$ and lax-normal functors to the category of profunctors (this has been discovered by Bénabou [3] and worked out in painful detail in [12]).

Here, we gloss over these abstract motivations and just give a very explicit definition of what this means, as what we need is just a particular case of the construction we worked out for guarded nets in [7].

Definition 10 (Internalization)**.** *Let* $(M, M^\sharp) \in \mathbf{Petri}^{\triangle}$ *be a hierarchical net. We define its* internalization*, denoted* $\int M^\sharp$*, as the following category:*

- *The objects of* $\int M^\sharp$ *are pairs* (X, x)*, where* X *is an object of* $\mathfrak{F}(M)$ *and* x *is an element of* $M^\sharp X$*. Concisely:*
$$\text{Obj} \int M^\sharp := \left\{ (X,x) \,\middle|\, (X \in \text{Obj}\, \mathfrak{F}(M)) \wedge (x \in M^\sharp X) \right\}.$$
- *A morphism from* (X, x) *to* (Y, y) *in* $\int M^\sharp$ *is a pair* (f, s) *where* $f \colon X \to Y$ *in* $\mathfrak{F}(M)$ *and* $s \in S_{M^\sharp f}$ *in the apex of the corresponding span connects* x *to* y*. Concisely:*
$$\begin{aligned}&\text{Hom}_{\int M^\sharp}[(X,x),(Y,y)] := \\ &:= \left\{ (f,s) \,\middle|\, (f \in \text{Hom}_{\mathfrak{F}(M)}[X,Y]) \wedge (s \in S_{M^\sharp f}) \wedge (M^\sharp f(s) = (x,y)) \right\}.\end{aligned}$$

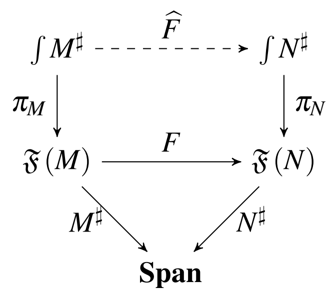
The category $\int N^\sharp$, called *the Grothendieck construction applied to* $N^\sharp$, produces a place for each element of the set we send a place to, and makes a transition for each path between these elements, as shown below:



Notice that in the picture above, on the left, each path between coloured dots is a triple (a,x,b) as in Remark 1. This amounts to promote every possible trace of the child net – together with a selection of initial and accepting states – to a transition in the parent net. This interpretation is justified by the following theorem, which we again proved in [7]:

Theorem 2. *Given any strict monoidal functor* $\mathfrak{F}(N) \xrightarrow{N^\sharp} \mathbf{Span}$*, the category* $\int N^\sharp$ *is symmetric strict monoidal, and free. Thus* $\int N^\sharp$ *can be written as* $\mathfrak{F}(M)$ *for some net* M*.*

Moreover, we obtain a projection functor $\int N^\sharp \to \mathfrak{F}(N)$ *which turns* $\int$ *into a functor, in that for each functor* $F : (M, M^\sharp) \to (N, N^\sharp)$ *there exists a functor* $\widehat{F}$ *making the following diagram commute:*



Theorem 2 defines a functor $\mathbf{Petri}^{\mathbf{Span}} \to \mathbf{FSSMC}$, the category of FSSMCs and strict monoidal functors between them. As $\mathbf{Petri}^{\blacktriangle}$ is a subcategory of $\mathbf{Petri}^{\mathbf{Span}}$, we can immediately restrict Theorem 2 to hierarchical nets. A net in the form $\int N^\sharp$ for some hiearchic net $(N, N^\sharp)$ is called the *internal categorical semantics for N* (compare this with Definition 8, which we called *external*).

Remark 3. *Notice how internalization is* very *different from just copy-pasting a child net in place of a transition in the parent net as we discussed in Section 3. Here, each* execution *of the child net is promoted to a transition, preserving the atomicity requirement of transitions in the parent net.*

Clearly, now we can define hiearchic nets with a level of hiearchy higher than 2 by just mapping a generator f of the parent net to a span where N_f is in the form $\int N^\sharp$ for some other hierarchical net N, and the process can be recursively applied any finite number of times, for each transition.

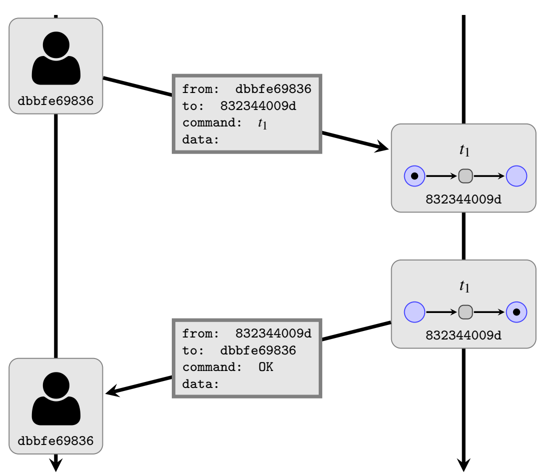
7 Engineering perspective

We deem wise to spend a few words on why we consider this way of doing things advantageous from an applicative point of view. Petri nets have been considered as a possible way of producing software for a long time, with some startups even using them as a central tool in their product offer [18]. Providing some form of hierarchical calling is absolutely needed to make the idea of “Petri nets as a programming language/general purpose design tool” practical.

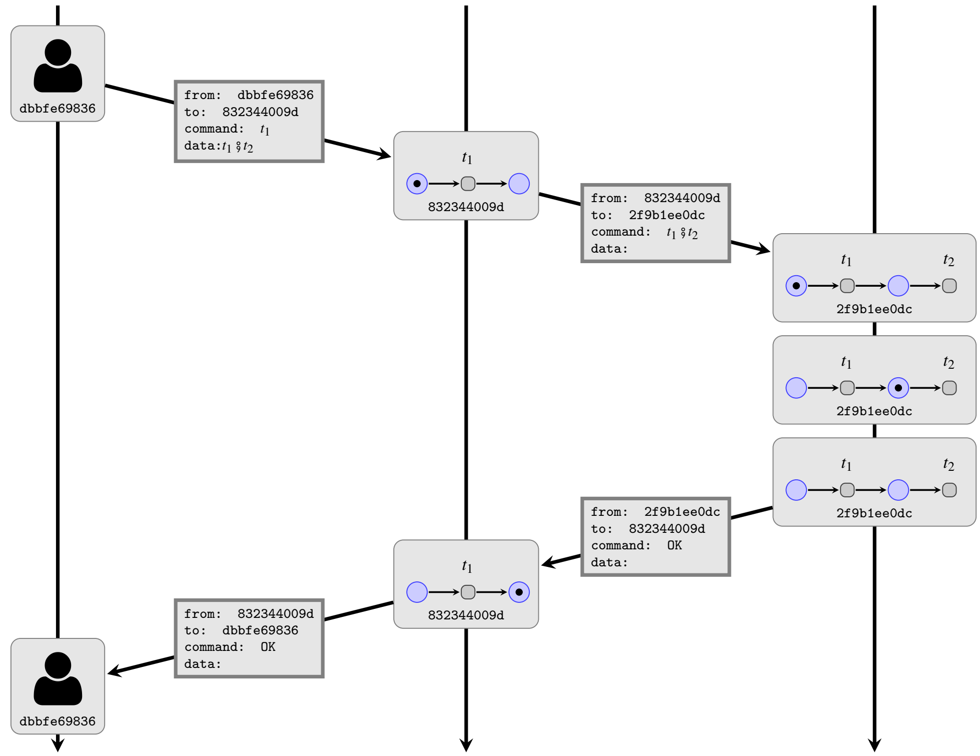
Our definition of hierarchy has the great advantage of not making hiearchic nets more expressive than Petri nets. If this seems like a downside, notice that a consequence of this is that decidability of any reachability-related question is exactly as for Petri nets, which is a great advantage from the point of view of model checking. Internalization, moreover, allows to compile hierarchical nets back to Petri nets, so that we can use already widespread tools for reachability checking [19] without having necessarily to focus on producing new ones.

Moreover, and actually more importantly, our span formalism works really well in modelling net behavior in a distributed setting. The parent and children nets may exist on different machines, and are not required to be engineered in a monolithic fashion.

To better understand this, imagine an infrastructure where each Petri net is considered as a piece of data with its own address (as it would be, for instance, if we were to implement nets as smart contracts on a blockchain). The way of operating Petri nets in this formalism is *transactional*: A user sends a message consisting of a net address, the transaction the user intends to fire, and some transaction data. The infrastructure replies affirmatively or negatively if the transaction can be fired, which amounts to accept or reject the transaction. Clearly this is particularly suitable for blockchain-related contexts and is how applications such as [17] implement Petri nets in their services.



From this point of view, a hierarchical net would work exactly as a standard Petri net, with the exception that in sending a transaction to the parent net the user has also to specify, in the transaction data, a valid execution of the child net corresponding to the firing transition.



Again, from a smart contract standpoint, this means that the smart contract corresponding to the parent net will call the contract corresponding to the child net with some execution data, and will respond affirmatively to the user only if the generated call resolves positively.

All the possible ways of executing the contracts above form a category, which is obtaining by internalizing the hierarchical net corresponding to them via Theorem 2.

Internalized categories being free, they are presented by Petri nets, which we can feed to any mainstream model checker. Now, all sorts of questions about liveness and interaction of the contracts above can be analyzed by model-checking the corresponding internalized net.

This provides an easy way to analyze complex contract interaction, relying on tools that have been debugged and computationally optimized for decades.

8 Conclusion and future work

In this work, we showed how a formalism for guarded nets already worked out in [7] can be used to define the categorical semantics of some particular flavor of hierarchical nets, which works particularly well from a model-checking and distributed-implementation point of view. Our effort is again part of a more ample project focusing on characterizing the categorical semantics of extensions of Petri nets by studying functors from FSSMCs to spans [6, 5].

As a direction of future work, we would like to obtain a cleaner way of describing recursively hierarchical nets. In this work, we relied on the Grothendieck construction to internalize a hierarchical net, so that we could use hierarchical nets as children of some other another parent net, recursively. This feels a bit like throwing all the carefully-typed information that the external semantics gives into the same bucket, and as such it is a bit unsatisfactory. Ideally, we would like to get a fully external semantics for recursively hierarchical nets, and generalize the internalization result to this case.

Another obvious direction of future work is implementing the findings hereby presented, maybe relying on some formally verified implementation of category theory such as [9].

Acknowledgements The first author was supported by the project MIUR PRIN 2017FTXR7S "IT-MaTTerS" and by the Independent Ethvestigator Program.

The third author was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

A video presentation of this paper can be found on Youtube at 4v5v8tgmiUM.

References

[1] John C. Baez and Jade Master. "Open Petri Nets". In: *Mathematical Structures in Computer Science* 30.3 (Mar. 2020), pp. 314–341. DOI: 10.1017/S0960129520000043. arXiv: 1808.05415 (cit. on p. 1).

[2] John C. Baez et al. *Categories of Nets*. Jan. 11, 2021. arXiv: 2101.04238 [cs, math]. URL: http://arxiv.org/abs/2101.04238 (visited on Jan. 13, 2021) (cit. on p. 2).

[3] Jean Bénabou. "Introduction to Bicategories". In: J. Bénabou et al. *Reports of the Midwest Category Seminar*. Vol. 47. Berlin, Heidelberg: Springer Berlin Heidelberg, 1967, pp. 1–77. DOI: 10.1007/BFb0074299 (cit. on p. 7).

[4] Fabrizio Genovese and Jelle Herold. "Executions in (Semi-)Integer Petri Nets Are Compact Closed Categories". In: *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 31, 2019), pp. 127–144. DOI: 10.4204/EPTCS.287.7 (cit. on p. 2).

[5] Fabrizio Genovese, Fosco Loregian, and Daniele Palombi. *A Categorical Semantics for Bounded Petri Nets*. Jan. 22, 2021. arXiv: 2101.09100 [cs, math]. URL: http://arxiv.org/abs/2101.09100 (visited on Jan. 25, 2021) (cit. on pp. 1, 10).

[6] Fabrizio Genovese, Fosco Loregian, and Daniele Palombi. *Nets with Mana: A Framework for Chemical Reaction Modelling*. Jan. 15, 2021. arXiv: 2101.06234 [cs, math, q-bio]. URL: http://arxiv.org/abs/2101.06234 (visited on Jan. 25, 2021) (cit. on pp. 1, 4, 10).

[7] Fabrizio Genovese and David I. Spivak. "A Categorical Semantics for Guarded Petri Nets". In: *Graph Transformation*. Ed. by Fabio Gadducci and Timo Kehrer. Vol. 12150. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 57–74. DOI: 10.1007/978-3-030-51372-6_4 (cit. on pp. 1, 2, 4, 7, 8, 10).

[8] Fabrizio Genovese et al. *Computational Petri Nets: Adjunctions Considered Harmful*. Apr. 29, 2019. arXiv: 1904.12974 [cs, math]. URL: http://arxiv.org/abs/1904.12974 (visited on May 2, 2019) (cit. on pp. 1, 2).

[9] Fabrizio Genovese et al. *Idris-Ct: A Library to Do Category Theory in Idris*. Nov. 25, 2019. arXiv: 1912.06191 [cs, math]. URL: http://arxiv.org/abs/1912.06191 (visited on Dec. 16, 2019) (cit. on pp. 1, 11).

[10] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. DOI: 10.1007/b95112 (cit. on p. 3).

[11] Michael Köhler-Bußmeier. "A Survey of Decidability Results for Elementary Object Systems". In: *Fundamenta Informaticae* 1 (2014), pp. 99–123. DOI: 10.3233/FI-2014-983 (cit. on p. 3).

[12] Fosco Loregian. *Coend Calculus*. Dec. 21, 2019. arXiv: 1501.02503 [math]. URL: http://arxiv.org/abs/1501.02503 (visited on Sept. 29, 2020) (cit. on p. 7).

[13] Jade Master. "Petri Nets Based on Lawvere Theories". In: *Mathematical Structures in Computer Science* 30.7 (Aug. 2020), pp. 833–864. DOI: 10.1017/S0960129520000262. arXiv: 1904.09091 (cit. on pp. 1, 2).

[14] José Meseguer and Ugo Montanari. "Petri Nets Are Monoids". In: *Information and Computation* 88.2 (Oct. 1990), pp. 105–155. DOI: 10.1016/0890-5401(90)90013-8 (cit. on pp. 1, 2).

[15] D. Pavlović and S. Abramsky. "Specifying Interaction Categories". In: *Category Theory and Computer Science*. Ed. by Eugenio Moggi and Giuseppe Rosolini. Red. by Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen. Vol. 1290. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 147–158. DOI: 10.1007/BFb0026986 (cit. on pp. 1, 7).

[16] Vladimiro Sassone. "On the Category of Petri Net Computations". In: *TAPSOFT '95: Theory and Practice of Software Development*. Ed. by Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach. Red. by Gerhard Goos, Juris Hartmanis, and Jan Leeuwen. Vol. 915. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 334–348. DOI: 10.1007/3-540-59293-8_205 (cit. on p. 2).

[17] Statebox Team. *Statebox, Compositional Diagrammatic Programming Language*. 2017. URL: https://statebox.org (cit. on p. 9).

[18] Statebox Team. *The Mathematical Specification of the Statebox Language*. June 18, 2019. arXiv: 1906.07629 [cs, math]. URL: http://arxiv.org/abs/1906.07629 (visited on June 28, 2019) (cit. on p. 8).

[19] University of Torino. *GreatSPN Github Page*. 2018. URL: https://github.com/greatspn/SOURCES (cit. on p. 9).

[20] Fabio Zanasi. *Interacting Hopf Algebras: The Theory of Linear Systems*. May 4, 2018. arXiv: 1805.03032 [cs, math]. URL: http://arxiv.org/abs/1805.03032 (visited on Jan. 3, 2021) (cit. on p. 6).