On Nominal Regular Languages with Binders*

Alexander Kurz, Tomoyuki Suzuki**, and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

Abstract. We investigate regular languages on infinite alphabets where words may contain binders on names. To this end, classical regular expressions and automata are extended with binders. We prove the equivalence between finite automata on binders and regular expressions with binders and investigate closure properties and complementation of regular languages with binders.

1 Introduction

Automata over infinite alphabets have been receiving an increasing amount of attention, see e.g. [16,22,4,26]. In these approaches, the countably infinite alphabet $\mathcal{N}$ can be considered as a set of 'names', which can be tested only for equality.

Typically, in this context languages of interest such as

$$\mathcal{L} = \{n_1 \dots n_k \in \mathcal{N}^* \mid \exists j > i \,.\, n_i = n_j\}$$

from [16] are invariant under name-permutations: If e.g. nmn is in the language, then so is $n'mn' = (n\,n') \cdot nmn$, where $(n\,n') \cdot nmn$ stands for the application of the transposition $(n\,n')$ to the word nmn. This suggests to think of names as being bound and languages to be closed under α-equivalence.

On the other hand, we may fix a name n_0 and consider the language

$$\mathcal{L}_{n_0} = \{n_0 n_1 \dots n_k \in \mathcal{N}^* \mid \exists j > i > 0 \,.\, n_i = n_j\}$$

so that we can think of n_0 as a free name and of $n_1, \dots, n_k$ as bound names. This suggests to study not only words over names, but also words which contain binders and allow us to make explicit the distinction between bound and free names. For example, we might then model $\mathcal{L}_{n_0}$ by a regular expression

$$n_0 \langle n. \langle m.m \rangle^* \, n \, \langle k.k \rangle^* \, n \rangle \tag{1}$$

where $\langle n.e \rangle$ binds n in e and, in the reading above, $\langle n.n \rangle^*$ is interpreted as $\mathcal{N}^*$.

In this paper, we consider languages with explicit binders on names in words and study regular expressions such as (1) together with the associated notion of finite automata. We prove a Kleene style theorem relating finite automata and regular expressions (cf. § 4) and show that regular languages are closed under intersection, union, and *resource-sensitive complement* (defined in § 5).

* This work has been partially supported by the Aut. Region of Sardinia under grant L.R.7/2007 CRP2-120 (Project TESLA).

** The author's PhD research was supported by the Yoshida Scholarship Foundation.

L. Birkedal (Ed.): FOSSACS 2012, LNCS 7213, pp. 255–269, 2012.

Regular expressions for words with binders could be used for the design and analysis of programming languages (as in [23] or [21]). For instance, to check that a 'variable' (i.e. name) is declared before it is used, consider the finite set of 'letters' $\mathcal{S} = \{\mathtt{DECL}, \mathtt{USE}\}$ and the nominal regular expression on the alphabet $\mathcal{N} \cup \mathcal{S}$

$$e_1 \, \langle n.\mathtt{DECL}\, n\, e_2\, \mathtt{USE}\, n\, e_3 \rangle^*$$

where with $n \in \mathcal{N}$ and e being some other regular expression not involving n.

Another motivation for words with binders comes from verification and testing. To this aim, in § 2 we consider a scenario based on transactional computations to show how regular languages with binders can suitably represent computations that classical regular languages can only approximate. More precisely, we give examples throughout the paper that illustrate how the naming policies used to implement transactional behaviours can be captured in terms of regular languages with binders. Then our Kleene theorem yields in this context an automata that recognises the language and our resource-sensitive complement operation can be used to obtain a transactional monitor, namely an automaton that recognises anomalous transactional behaviours.

Related Work. Automata on words with binders already appear in [24] in the study of the λ-calculus. In this paper we begin the systematic study of words with binders from the point of view of the classical theory of formal languages and automata. This builds on the recent convergence of three lines of research: languages over infinite alphabets, HD-automata, and nominal sets, as we will explain now in more detail. As emphasised above, the languages over infinite alphabets typically considered are *equivariant*, that is, they are invariant under *permutations*. On the other hand, history-dependent (HD) automata [19,18] have been developed in order to check π-calculus expressions for bisimilarity. This research highlighted that minimization requires to keep track of permutations of local names. It is also known that HD-automata are automata internal (in the sense of [3]) in the category of *named sets* [6,7]. Independently of the work on HD-automata, nominal sets [13] were introduced as a framework of doing mathematics in a set theory where every set comes equipped with a permutation action. In particular, in nominal sets, binders arise naturally as *name abstractions*. Later it was shown that the categories of nominal sets and named sets are equivalent [11,14].

Recently, these three developments have been coming together, see [8,26,4,12] and in particular [5]. In these works, the idea of automata and language theory internal in nominal sets takes shape, but, although binders are the *raison d'être* of nominal sets they are, so far, not present in the words themselves.

Structure of the Paper. In § 2 we describe a transactional scenario used throughout the paper. In § 3 we define regular languages with binders, nominal regular expressions, and automata on binders. § 4 and 5 contain the main technical contributions of the paper.

2 Representing Transactions

Our running example is centred around the notion of *nested transactions* which are paramount in information systems [27] because they feature data consistency in the presence of concurrent or distributed accesses. A transaction is a logically atomic computation made of several steps. A transaction either commits when all its steps are

successful or rolls partial computations back when failures occur before completion. Nested transactions are transactions that may possibly contain inner transactions so that the failure of an inner transaction is confined and does not affect outer transactions. For instance, let A, B, and C be basic activities subject to failure in the nested transaction

$$\texttt{beginTX}\ A\ ;\ \texttt{beginTX}\ B\ \texttt{endTX}\ ;\ C\ \texttt{endTX} \tag{2}$$

where C is executed even if B fails and the outer transaction is successful if A and C do not fail; on the contrary, a failure of C would require that also B is rolled-back.

Inner transactions can abort outer ones. This is typically implemented by using *transaction identities* that allow inner transactions to invoke abort operations on outer transactions. Using identities transaction (2), becomes

$$\texttt{beginTX}_2\ A\ ;\ \texttt{beginTX}_1\ B\ \texttt{endTX}_1\ ;\ C\ \texttt{endTX}_2 \tag{3}$$

and B can execute an abort operation (say $\texttt{abt}_2$) that makes transaction (3) fail.

We will consider *i-bound* nested transactions, namely transactions that can be nested only up to a fixed level i. To characterise correct executions of bound nested transactions, one could think of using regular expressions. For instance, take the alphabet

$$T \quad = \quad \{\mathtt{s}_1,\ldots,\mathtt{s}_h\} \cup \{\checkmark,\times\} \cup \{[^j,]^j,\mathtt{cmt}^j,\mathtt{abt}^j\}_{1\le j\le i}$$

where symbols $\mathtt{s}_i$ represent basic activities and the others denote success ($\checkmark$), failure ($\times$), and — for each possible nesting level — begin ($[^j$), end ($]^j$), and the intention to commit ($\mathtt{cmt}^j$) or abort ($\mathtt{abt}^j$). Consider the following regular expressions (4), (5), and (6) on T where, for simplicity, we examine 2-bound transactions:

$$e_0 = \Big(\sum_{i=1}^{h} \mathtt{s}_i\Big)^* \tag{4}$$

$$e_1 = \Big(e_0\ +\ [^1\ e_0\ \mathtt{cmt}^1\ ]^1\checkmark\ +\ [^1\ e_0\ \mathtt{abt}^1\ ]^1\ \times\Big)^* \tag{5}$$

$$e_2 = \Big(e_0\ +\ [^2\ e_1\ \mathtt{cmt}^2\ ]^2\checkmark\ +\ [^2\ e_1\ \mathtt{abt}^2\ ]^2\ \times\ +\ [^2\ e_1\ [^1\ e_0\ \mathtt{abt}^2\ ]^1\ ]^2\ \times\Big)^* \tag{6}$$

The language corresponding to e_2 characterises the correct executions of computations with transactions nested up to level two. Therefore, the automaton recognising the complement of the language of e_2 can be used as monitor of such transactions.

Although the expressions (4), (5), and (6) correctly capture the structure of correct executions of our transactions (balanced parenthesis up to level 2), a main drawback is that they do not suitably represent identities of transactions. For example,

$$\texttt{beginTX}_1\ \mathtt{s}_1\ \texttt{endTX}_1\ \ldots\ \texttt{beginTX}_k\ \mathtt{s}_1\ \texttt{endTX}_k \tag{7}$$

where k is unbound and all the identifiers $\texttt{beginTX}_i$ are pairwise distinct (and similarly for $\texttt{endTX}_i$), represents the sequential execution of k (non-nested) transactions. Translated in our alphabet T, computation (7) becomes the word $[^1\ \mathtt{s}_1\ ]^1\checkmark\ \ldots\ [^1\ \mathtt{s}_1\ ]^1\checkmark$, where the identities of the transactions vanish. Note that the alternative $[^1\ \mathtt{s}_1\ ]^1\checkmark\ \ldots\ [^k\ \mathtt{s}_1\ ]^k\checkmark$ requires an infinite alphabet because k is unbound.

We define nominal regular languages below and give a nominal regular expression that captures computations like (7) (cf. Example 1). We provide a class of automata with binders able to accept such language (cf. Example 2).

3 Languages, Automata and Regular Expressions, with Binders

In this section, we introduce languages, automata and regular expressions to present examples as above in a uniform and formal way.

Languages. The main idea is to handle *local names* by explicitly denoting the *binding scopes* to express locality. A binding scope takes the form $\langle\!\langle n.\cdots\rangle\!\rangle$ and represents the fact that the name n is bound between the scope delimiters $\langle\!\langle$ and $\rangle\!\rangle$. For instance, the word $\langle\!\langle n.n\ m\ n\rangle\!\rangle$ has the occurrences of n bound while m is not affected by the binder (it occurs *free* in the word). Consequently, we consider words up to α-renaming for bound names, e.g. $\langle\!\langle n.n\ m\ n\rangle\!\rangle$ is identified with $\langle\!\langle n'.n'\ m\ n'\rangle\!\rangle$ for any $n' \neq m$.

Now, let $\mathcal{N}$ be a countably infinite set (of 'names') and $\mathcal{S}$ a finite set (of 'letters'). We define *words* w according to

$$w ::= \varepsilon \mid n \mid s \mid w \circ w \mid \langle\!\langle n.w\rangle\!\rangle$$

where n ranges over $\mathcal{N}$ and s over $\mathcal{S}$. We do not consider equalities on words other than α-renaming and, as in the classical case, the monoidal laws of composition. Namely, every word is taken up to α-renaming and the concatenation operation $\circ$ is associative and has the empty word ε as the neutral element. We often write $w\ v$ for $w \circ v$. We call a set of words (closed under α-renaming) a *nominal language*, or simply a *language*.

The occurrence in a word w of $n \in \mathcal{N}$ is *bound* (resp. *free*) if it is (resp. not) in the scope of a binder $\langle\!\langle n.\_\rangle\!\rangle$.

Regular Expressions. We define regular expressions with binders, or *nominal regular expressions* via the grammar

$$\mathsf{ne} ::= 1 \mid 0 \mid n \mid s \mid \mathsf{ne} \circ \mathsf{ne} \mid \mathsf{ne} + \mathsf{ne} \mid \langle n.\mathsf{ne}\rangle \mid \mathsf{ne}^*$$

An occurrence of a name n in a nominal regular expression ne, is *bound* if it is in the scope of a binder $\langle n.\_\rangle$, otherwise it is *free*; accordingly, we say that n is a bound (resp. free) name of ne if there are bound (resp. free) occurrences of n in ne and we let $FN(\mathsf{ne})$ be the set of free names in ne (since ne is a finite expression, $FN(\mathsf{ne})$ is finite).

Example 1. We describe the nominal regular expression addressing the problem from § 2(7)*. Let* $\mathcal{S}_{tx} = \{\checkmark, \times, \mathtt{cmt}, \mathtt{abt}\} \cup \{\mathtt{s}_1, \ldots, \mathtt{s}_h\}$ *and* $n_1, n_2 \in \mathcal{N}$ *be distinct. Define*

$$\mathsf{ne}_1 = \Big(e_0 \ + \ \langle n_1.\ e_0\ \mathtt{cmt}\ n_1\rangle \checkmark + \langle n_1.\ e_0\ \mathtt{abt}\ n_1\rangle \ \times\Big)^* \tag{8}$$

$$\mathsf{ne}_2 = \Big(e_0 \ + \ \langle n_2.\ \mathsf{ne}_1\ \mathtt{cmt}\ n_2\rangle \checkmark + \langle n_2.\ \mathsf{ne}_1\ \mathtt{abt}\ n_2\rangle \ \times \ + \ \langle n_2.\ \mathsf{ne}_1\ \langle n_1.\ e_0\ \mathtt{abt}\ n_2\rangle\rangle \ \times\Big)^* \tag{9}$$

where e_0 *is defined in* (4)*. The above equations are rather similar to* (5)*, and* (6)*; however, in* (8) *and* (9)*, the binders delimit the scope of* n_1 *and* n_2 *and correspond to the beginning and ending of transactions.*

In Example 1, identities of transactions are modelled as bound names. Computations of the form (7) are captured by ne_1 that simply requires to re-bind n_1 to $\mathtt{beginTX}_{i+1}$

after it has been bound to $\texttt{beginTX}_i$. This is made more precise by considering the interpretation of nominal regular expressions below.

The nominal language $L(\mathsf{ne})$ of a nominal regular expression ne, is defined as

$$L(1) \stackrel{\text{def}}{=} \{\varepsilon\} \qquad L(0) \stackrel{\text{def}}{=} \emptyset \qquad L(n) \stackrel{\text{def}}{=} \{n\} \qquad L(s) \stackrel{\text{def}}{=} \{s\}$$

$$L(\mathsf{ne}_1 + \mathsf{ne}_2) \stackrel{\text{def}}{=} L(\mathsf{ne}_1) \cup L(\mathsf{ne}_2) \qquad L(\langle n.\mathsf{ne}\rangle) \stackrel{\text{def}}{=} \{\langle\!\langle n.w \rangle\!\rangle \mid w \in L(\mathsf{ne})\}$$

$$L(\mathsf{ne}_1 \circ \mathsf{ne}_2) \stackrel{\text{def}}{=} L(\mathsf{ne}_1) \circ L(\mathsf{ne}_2) = \{w \circ v \mid w \in L(\mathsf{ne}_1), v \in L(\mathsf{ne}_2)\}$$

$$L(\mathsf{ne}^*) \stackrel{\text{def}}{=} \bigcup_{j \in \mathbb{N}} L(\mathsf{ne})^j, \text{ where } L(\mathsf{ne})^j \stackrel{\text{def}}{=} \begin{cases} \{\varepsilon\} & j = 0 \\ L(\mathsf{ne}) \circ L(\mathsf{ne})^{j-1} & j \neq 0 \end{cases}$$

A language of the form $L(\mathsf{ne})$ is called a *nominal regular language*.

Automata. To describe a mechanism to handle *local names* and binders, we let $\mathbb{N}$ denote the set of natural numbers and define $\underline{i} = \{1, \ldots, i\}$ for each $i \in \mathbb{N}$. We consider sets (of states) Q paired with a map $\|\_\| : Q \to \mathbb{N}$ and define the *local registers of* $q \in Q$ to be $\underline{\|q\|}$. Definition 2 below explains how registers store names via maps $\sigma : \underline{\|q\|} \to \mathcal{N}$.

Definition 1. *Let $\mathcal{N}_{fin} \subseteq \mathcal{N}$ be a finite set of names. An* automaton on binders over $\mathcal{S}$ and $\mathcal{N}_{fin}$ *— an* $(\mathcal{S}, \mathcal{N}_{fin})$-automaton *for short — is a tuple $\mathcal{H} = \langle Q, q_0, F, tr\rangle$ such that*

- *Q is a finite set (of states) equipped with a map $\|\_\| : Q \to \mathbb{N}$*
- *$q_0 \in Q$ is the* initial state *and $\|q_0\| = 0$*
- *$F \subseteq Q$ is the set of* final states *and $\|q\| = 0$ for each $q \in F$*
- *for each $q \in Q$ and $\alpha \in I^{\mathcal{N}_{fin}}(q) \cup \{\varepsilon\}$ — where $I^{\mathcal{N}_{fin}}(q) \stackrel{\text{def}}{=} \mathcal{S} \cup \{\langle\!\langle, \rangle\!\rangle\} \cup \underline{\|q\|} \cup \mathcal{N}_{fin}$ is the set of possible inputs on q — we have a set $tr(q, \alpha) \subseteq Q$ of 'successor states'; for all $q' \in tr(q, \alpha)$ the following must hold:*

$$\begin{aligned} \alpha = \langle\!\langle \quad &\Longrightarrow \quad \|q'\| = \|q\| + 1 \\ \alpha = \rangle\!\rangle \quad &\Longrightarrow \quad \|q'\| = \|q\| - 1 \\ \alpha \in I^{\mathcal{N}_{fin}}(q) \setminus \{\langle\!\langle, \rangle\!\rangle\} \text{ or } \alpha = \varepsilon \quad &\Longrightarrow \quad \|q'\| = \|q\| \end{aligned}$$

An α-transition is a triple (q, α, q') such that $q' \in tr(q, \alpha)$.

$\mathcal{H}$ is deterministic *if, for each $q \in Q$,*

$$\begin{cases} |tr(q,\alpha)| = 0, & \text{if } (\alpha = \langle\!\langle \text{ and } \|q\| = \max\{\|q'\| \mid q' \in Q\}) \text{ or } (\alpha = \rangle\!\rangle \text{ and } \|q\| = 0) \\ |tr(q,\alpha)| = 1, & \text{otherwise.} \end{cases}$$

The condition $\|q\| = 0$ for each $q \in F \cup \{q_0\}$ in Definition 1 can be removed at the cost of making the presentation technically more complex.

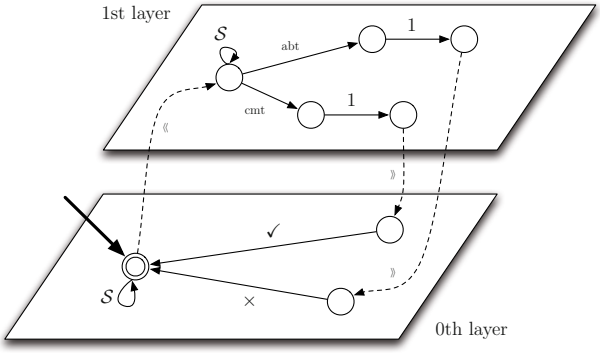
As in the classical case, we say that a $(\mathcal{S}, \mathcal{N}_{fin})$-automaton is *accessible* when all its states are reachable from the initial state. We tacitly assume that all $(\mathcal{S}, \mathcal{N}_{fin})$-automata in the current paper are accessible.

Fact 1. *For any $(\mathcal{S}, \mathcal{N}_{fin})$-automaton $\mathcal{H} = \langle Q, q_0, F, tr\rangle$ and $q \in Q$ we have that $\|q\| = 0$ implies $tr(q, \rangle\!\rangle) = \emptyset$ and that $\|q\| = \max_{q' \in Q} \|q'\|$ implies $tr(q, \langle\!\langle) = \emptyset$.*

The notion of determinism in Definition 1 is slightly different from the classical one because it must consider the constraints between the registers of source and target states of transitions. We shall come back to this in § 5.

Example 2 below exhibits an $(\mathcal{S},\mathcal{N}_{fin})$-automaton for the nominal regular expression ne_1 in Example 1. Instead of the formal definition, we introduce a more appealing graphical notation, anticipating the notion of *layer* defined in § 4.

Example 2. Let $\mathcal{S}_{tx}$ as defined in Example 1. The $(\mathcal{S}_{tx},\emptyset)$-automaton corresponding to ne_1 can be depicted as



which has a unique initial and final state (the circled one on the 0th layer).

In the figure of Example 2, dashed transitions denote $\ll$- and $\gg$-transitions. The $\ll$-transition goes from a state on the 0th layer to a state on the 1st layer, whereas the two $\gg$-transitions go in the opposite direction. Also note that within each layer the picture shows essentially a classical automaton. This is typical for $(\mathcal{S},\mathcal{N}_{fin})$-automata, see § 4.

Let us fix an $(\mathcal{S},\mathcal{N}_{fin})$-automaton

$$\mathcal{H} \stackrel{\text{def}}{=} \langle Q, q_0, F, tr\rangle. \tag{10}$$

and define how $\mathcal{H}$ processes words with free names in $\mathcal{N}_{fin}$. Hereafter, we denote the image of a map σ by $Im(\sigma)$ and the empty map by $\emptyset$.

A *configuration* of $\mathcal{H}$ is a tuple $\langle q,w,\sigma\rangle$ consisting of a state q, a word w whose free names are in $\mathcal{N}_{fin}\cup Im(\sigma)$, and a map $\sigma\colon \underline{\|q\|} \to \mathcal{N}$. We call a configuration $\langle q,w,\sigma\rangle$ *initial* if $q=q_0$, w is a word whose free names are in $\mathcal{N}_{fin}$, and $\sigma=\emptyset$; we call $\langle q,w,\sigma\rangle$ *accepting* if $q\in F$, $w=\varepsilon$, and $\sigma=\emptyset$.

Definition 2. *Given $q,q'\in Q$ and two configurations $t=\langle q,w,\sigma\rangle$ and $t'=\langle q',w',\sigma'\rangle$, $\mathcal{H}$ as in (10)* moves from *t* to *t' (written $t \stackrel{\mathcal{H}}{\to} t'$) if there is $\alpha\in\mathcal{N}\cup\mathbb{N}\cup\mathcal{S}\cup\{\ll,\gg,\varepsilon\}$ such that $q'\in tr(q,\alpha)$ and*

$$\begin{cases}
\alpha\in\underline{\|q\|}, & w=\sigma(\alpha)\,w', & \sigma'=\sigma & \text{and} \quad \forall i>\alpha.\sigma(\alpha)\neq\sigma(i)\\
\alpha\in\underline{\mathcal{N}_{fin}}\setminus Im(\sigma), & w=\alpha\,w', & \text{and } \sigma'=\sigma & \\
\alpha\in\mathcal{S}, & w=\alpha\,w', & \text{and } \sigma'=\sigma & \\
\alpha=\varepsilon, & w=w', & \text{and } \sigma'=\sigma & \\
\alpha=\gg, & w=\gg w', & \text{and } \sigma'=\sigma_{|\underline{\|q\|}'} & \\
\alpha=\ll, & w=\ll n.w', & \text{and } \sigma'=\sigma[\underline{\|q'\|}\mapsto n] &
\end{cases}$$

where $\sigma[\|q'\| \mapsto n]$ *extends* σ *by allocating the maximum index in* $\underline{\|q'\|}$ *to n.*

The set $reach_{\mathcal{H}}(t)$ *of states reached by* $\mathcal{H}$ *from the configuration t is defined as*

$$reach_{\mathcal{H}}(t) \stackrel{\text{def}}{=} \begin{cases} \{q\} & \text{if } t = \langle q, \varepsilon, \sigma \rangle \\ \bigcup_{t \xrightarrow{\mathcal{H}} t'} reach_{\mathcal{H}}(t') & \text{otherwise} \end{cases}$$

A run of $\mathcal{H}$ on a word *w is a sequence of moves of* $\mathcal{H}$ *from* $\langle q_0, w, \emptyset \rangle$.

Definition 3. *The* $(\mathcal{S}, \mathcal{N}_{fin})$*-automaton* $\mathcal{H}$ *in (10)* accepts *(or* recognises*) a word w if* $F \cap reach_{\mathcal{H}}(\langle q_0, w, \emptyset \rangle) \neq \emptyset$. *The* language of $\mathcal{H}$ *is the set* $\mathcal{L}_{\mathcal{H}}$ *of words accepted by* $\mathcal{H}$.

Example 3. It is straightforward to observe that the $(\mathcal{S}_{tx}, \emptyset)$*-automaton in Example 2 accepts* $L(\mathsf{ne}_1)$. *The only interesting steps are from a configuration where the word starts with a binder, say* $\langle\!(n.w)\!\rangle$. *In our example, the automaton consumes the binder only if it is in the initial/final state; in this case, the (unique) register of the target state on the* 1*st layer is mapped to n and used in the transitions on the* 1*st layer. Observe that, if the right-most states on the* 1*st layer are reached by consuming w, then the automaton can "deallocate" n and possibly reach the final state.*

A direct consequence of Definitions 2 and 3 is the following proposition.

Proposition 1. *If* $\mathcal{H}$ *accepts w and* w' *is* α*-equivalent to w then* $\mathcal{H}$ *accepts* w'.

Remark 1. The automata in Definition 1 can be envisaged either as an instantiation of *basic history-dependent* automata [19] or as a variant of *finite-memory automata* [16]. In fact, the constraint on local names imposed in Definition 1 allows us to treat names as "global" (as done in finite-memory automata). More precisely, the semantics of each index is uniformly fixed through our automata, once it has been allocated. A main difference wrt basic history-dependent and finite-memory automata though, is the "stack discipline" imposed by $\langle\!(-$ and $-\!\rangle$-transitions.

4 A Kleene Theorem

This section gives the details of the equivalence, in the setting with binders, of finite automata and regular expressions. The main results can be summarised as follows.

Theorem 1. *For each* $(\mathcal{S}, \mathcal{N}_{fin})$*-automaton* $\mathcal{H}$*, there exists a deterministic* $(\mathcal{S}, \mathcal{N}_{fin})$*-automaton which recognises the same language as* $\mathcal{H}$.

Theorem 2. *Every language recognised by an* $(\mathcal{S}, \mathcal{N}_{fin})$*-automaton is representable by a nominal regular expression. Conversely, every language represented by a nominal regular expression* ne *is acceptable by an* $(\mathcal{S}, FN(\mathsf{ne}))$*-automaton.*

The proofs and constructions are obtained by extending the techniques of classical automata theory (see e.g. [15]) layer-wise: Given $\mathcal{H} = \langle Q, q_0, F, tr \rangle$ as in (10), $Q^i \stackrel{\text{def}}{=} \{q \in Q \mid \|q\| = i\}$ is the i-th *layer* of $\mathcal{H}$. Each layer of $\mathcal{H}$ is very much like a classical automaton if all $\langle\!(-$ and $-\!\rangle$-transitions are ignored. In fact, the only way to move from

layer i to $i+1$ is to read a $\langle\!\langle$ along a $\langle\!\langle$-transition; vice-versa, moving from layer $i+1$ to i is possible only by reading a $\rangle\!\rangle$ along a $\rangle\!\rangle$-transition.

From $(\mathcal{S},\mathcal{N}_{fin})$-Automata to Regular Expressions. The first step to prove Theorem 1 is to construct a deterministic $(\mathcal{S},\mathcal{N}_{fin})$-automaton for each $(\mathcal{S},\mathcal{N}_{fin})$-automaton. Given an $(\mathcal{S},\mathcal{N}_{fin})$-automaton $\mathcal{H}$, we first remove all ε-transitions. Note that ε-transitions are not allowed to connect states on different layers. For the ε-free non-deterministic $(\mathcal{S},\mathcal{N}_{fin})$-automaton, we take *the powerset construction* for each layer, and make all layers deterministic except $\langle\!\langle$- and $\rangle\!\rangle$-transitions. Finally, we define $\langle\!\langle$-transitions and $\rangle\!\rangle$-transitions in a deterministic way: For each subset $Q' \subseteq Q^i$, we let

$$tr(Q',\langle\!\langle) \stackrel{\text{def}}{=} \bigcup_{q\in Q'} tr(q,\langle\!\langle) \qquad \text{and} \qquad tr(Q',\rangle\!\rangle) \stackrel{\text{def}}{=} \bigcup_{q\in Q'} tr(q,\rangle\!\rangle)$$

This construction allows us to claim Theorem 1.

There are two main reasons for applying the powerset construction layer-wise rather than to the whole automaton. A technical reason is that the definition of the function $\|\_\|$ on sets of states taken from different layers could not be given in a consistent way. Secondly, since only $\langle\!\langle$- and $\rangle\!\rangle$-transitions can move between layers, each layer must have a "sink" state (i.e. the empty set of states) to allow for transitions that reject words.

The following proposition yields one direction of Theorem 2.

Proposition 2. *For any deterministic $(\mathcal{S},\mathcal{N}_{fin})$-automaton $\mathcal{H}$ there is a nominal regular expression $\mathsf{ne}_{\mathcal{H}}$ such that $L(\mathsf{ne}_{\mathcal{H}})$ is the language recognised by $\mathcal{H}$.*

Proof (Sketch). Take $\mathcal{H}$ as in (10) to be deterministic; Q can be decomposed into

$$Q^0 = \{q_1^0,\dots,q_{m_0}^0\}, \qquad Q^1 = \{q_1^1,\dots,q_{m_1}^1\}, \qquad \cdots \qquad Q^h = \{q_1^h,\dots,q_{m_h}^h\}$$

where $h = \max_{q\in Q}\|q\|$ is the highest layer of $\mathcal{H}$. Note that $q_0 \in Q^0$ and we assume that it is q_1^0. Let ${}^sR_{i,j}^k$ denote the set of paths from q_i^s to q_j^s which visit only states on layers higher than s or states $q_r^s \in Q^s$ with $r \le k$. Then, ${}^sR_{i,j}^k$ is defined on the highest layer h by

$${}^hR_{i,j}^0 \stackrel{\text{def}}{=} \{\alpha \mid q_j^h \in tr(q_i^h,\alpha)\} \cup E \qquad {}^hR_{i,j}^k \stackrel{\text{def}}{=} {}^hR_{i,k}^{k-1}\left({}^hR_{k,k}^{k-1}\right)^*{}^hR_{k,j}^{k-1} \cup {}^hR_{i,j}^{k-1}$$

where, in the first clause, $E = \{\varepsilon\}$ if $i = j$ and $E = \emptyset$ if $i \neq j$ and, for layer $s < h$, letting $\Gamma_{i,j}^s \stackrel{\text{def}}{=} \{(i',j') \mid q_{i'}^{s+1} \in tr(q_i^s,\langle\!\langle) \wedge q_j^s \in tr(q_{j'}^{s+1},\rangle\!\rangle)\}$, by

$${}^sR_{i,j}^0 \stackrel{\text{def}}{=} \{\alpha \mid q_j^s \in tr(q_i^s,\alpha)\} \cup \bigcup_{(i',j')\in\Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_{s+1}} \cup E$$

$${}^sR_{i,j}^k \stackrel{\text{def}}{=} {}^sR_{i,k}^{k-1}\left({}^sR_{k,k}^{k-1}\right)^*{}^sR_{k,j}^{k-1} \cup {}^sR_{i,j}^{k-1} \cup \bigcup_{(i',j')\in\Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_{s+1}}$$

where, in the first clause, $E = \{\varepsilon\}$ if $i = j$ and $E = \emptyset$ if $i \neq j$.

Hence, $\bigcup_{(i',j')\in\Gamma_{i,j}^s} {}^{s+1}R_{i',j'}^{m_s}$ is the collection of all paths from q_i^s to q_j^s visiting only states on the higher layers. Finally, we translate all paths from the initial state to final states into a nominal regular expression, but this is analogous to the classical theory. □

From Nominal Regular Expressions to $(\mathcal{S},\mathcal{N}_{fin})$-Automata. We now turn our attention to the construction of $(\mathcal{S},\mathcal{N}_{fin})$-automata from nominal regular expressions.

Proposition 3. *Given a nominal regular expression* ne*, there is an* $(\mathcal{S},FN(\mathsf{ne}))$*-automaton* $\mathcal{H}$ *which recognises* $L(\mathsf{ne})$.

We prove the above proposition by induction on the structure of ne. Let $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ denote the $(\mathcal{S},FN(\mathsf{ne}))$-automaton defined by the following construction.

We start with the constructions for the base cases.

- $\mathsf{ne}=1$: let $\mathcal{H}_{(\!|1|\!)}$ be $\langle\{q_0\},q_0,\{q_0\},tr\rangle$ where $\|q_0\|=0$ and $tr(q_0,\alpha)=\emptyset$ for each $\alpha\in I^{\mathcal{N}_{fin}}(q_0)$.
- $\mathsf{ne}=0$: let $\mathcal{H}_{(\!|0|\!)}$ be $\langle\{q_0\},q_0,\emptyset,tr\rangle$ where $\|q_0\|=0$ and $tr(q_0,\alpha)=\emptyset$ for each $\alpha\in I^{\mathcal{N}_{fin}}(q_0)$.
- $\mathsf{ne}=n$: let $\mathcal{H}_{(\!|n|\!)}$ be $\langle\{q_0,q_1\},q_0,\{q_1\},tr\rangle$ where, for $j\in\{0,1\}$ $\|q_j\|=0$ and

$$\begin{array}{ll} tr(q_0,n) & =\{q_1\}\\ tr(q_j,\alpha) & =\emptyset, \qquad \text{for } \alpha\in I^{\mathcal{N}_{fin}}(q_j) \text{ and } \alpha\neq n \text{ if } j=0.\end{array}$$

  Note that, as $FN(n)=\{n\}$, each state may have a transition with the label n.
- $\mathsf{ne}=s$: let $\mathcal{H}_{(\!|s|\!)}$ be $\langle\{q_0,q_1\},q_0,\{q_1\},tr\rangle$ where, for $j\in\{0,1\}$, $\|q_j\|=0$ and

$$\begin{array}{ll} tr(q_0,s) & =\{q_1\}\\ tr(q_j,\alpha) & =\emptyset, \qquad \text{for } \alpha\in I^{\mathcal{N}_{fin}}(q_j) \text{ and } \alpha\neq s \text{ if } j=0.\end{array}$$

Lemma 1. *$\mathcal{H}_{(\!|1|\!)}$, $\mathcal{H}_{(\!|0|\!)}$, $\mathcal{H}_{(\!|n|\!)}$ and $\mathcal{H}_{(\!|s|\!)}$ recognise, respectively, $L(1)$, $L(0)$, $L(n)$ and $L(s)$. Further, $\mathcal{H}_{(\!|1|\!)}$, $\mathcal{H}_{(\!|0|\!)}$ and $\mathcal{H}_{(\!|s|\!)}$ are $(\mathcal{S},\emptyset)$-automata, and $\mathcal{H}_{(\!|n|\!)}$ is an $(\mathcal{S},\{n\})$-automaton, i.e. $\mathcal{H}_{(\!|1|\!)}$, $\mathcal{H}_{(\!|0|\!)}$, $\mathcal{H}_{(\!|n|\!)}$ and $\mathcal{H}_{(\!|s|\!)}$ are all $(\mathcal{S},FN(\mathsf{ne}))$-automata.*

Union $\mathsf{ne}_1+\mathsf{ne}_2$: For $j\in\{1,2\}$, let $\mathcal{H}_{(\!|\mathsf{ne}_j|\!)}=\langle Q_j,q_{0,j},F_j,tr_j\rangle$ be an $(\mathcal{S},FN(\mathsf{ne}_j))$-automaton which recognises $L(\mathsf{ne}_j)$. The union $\mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)}$ is a tuple $\langle Q^+,q_0^+,F^+,tr^+\rangle$:

- $Q^+ \stackrel{\text{def}}{=} Q_1\uplus Q_2\uplus\{q_0^+\}$ (where $\uplus$ stands for disjoint union);
- q_0^+ is a new state with $\left\|q_0^+\right\|=0$;
- $F^+ \stackrel{\text{def}}{=} F_1\uplus F_2$;
- $tr^+(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q,\alpha), & \text{if } j\in\{1,2\},\ q\in Q_j, \text{ and } \alpha\in I_j^{\mathcal{N}_{fin}}(q)\\ \{q_{0,1},q_{0,2}\}, & \text{if } q=q_0^+ \text{ and } \alpha=\varepsilon\\ \emptyset, & \text{otherwise}\end{cases}$

where $I_1^{\mathcal{N}_{fin}}(q)$ is the set of possible inputs on q in $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)}$ and $I_2^{\mathcal{N}_{fin}}(q)$ is the set of possible inputs on q in $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)}$. Notice that possible inputs of free names are extended from $FN(\mathsf{ne}_1)$ or $FN(\mathsf{ne}_2)$ to $FN(\mathsf{ne}_1)\cup FN(\mathsf{ne}_2)$ in $\mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)}$, but the above definition of transitions says that each state q in Q_1 has no transition with labels in $FN(\mathsf{ne}_2)\setminus I_1^{\mathcal{N}_{fin}}(q)$, and each state q in Q_2 has no transition with labels in $FN(\mathsf{ne}_1)\setminus I_2^{\mathcal{N}_{fin}}(q)$.

Lemma 2. *$\mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)}$ is an $(\mathcal{S},FN(\mathsf{ne}_1+\mathsf{ne}_2))$-automaton recognising $L(\mathsf{ne}_1+\mathsf{ne}_2)$.*

Concatenation $\mathsf{ne}_1 \circ \mathsf{ne}_2$: For $j \in \{1,2\}$, let $\mathcal{H}_{(\!|\mathsf{ne}_j|\!)} = \langle Q_j, q_{0,j}, F_j, tr_j\rangle$ be an $(\mathcal{S},FN(\mathsf{ne}_j))$-automaton which recognises $L(\mathsf{ne}_j)$. The concatenation $\mathcal{H}_{(\!|\mathsf{ne}_1\circ\mathsf{ne}_2|\!)}$ is a tuple $\langle Q^\circ, q_0^\circ, F^\circ, tr^\circ\rangle$:

- $Q^\circ \stackrel{\text{def}}{=} Q_1 \uplus Q_2$;
- $q_0^\circ \stackrel{\text{def}}{=} q_{0,1}$;
- $F^\circ \stackrel{\text{def}}{=} F_2$;
- $tr^\circ(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q,\alpha), & \text{if } q \in Q_1 \setminus F_1, \text{ and } \alpha \in I_1^{\mathcal{N}_{fin}}(q) \\ tr_2(q,\alpha), & \text{if } q \in Q_2 \text{ and } \alpha \in I_2^{\mathcal{N}_{fin}}(q) \\ tr_1(q,\alpha) \cup \{q_{0,2}\}, & \text{if } q \in F_1 \text{ and } \alpha = \varepsilon \\ tr_1(q,\alpha), & \text{if } q \in F_1 \text{ and } \alpha \in I_1^{\mathcal{N}_{fin}}(q) \\ \emptyset, & \text{otherwise} \end{cases}$

Intuitively, we just add ε-transitions to $q_{0,2}$ from the final states of $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)}$.

Lemma 3. *$\mathcal{H}_{(\!|\mathsf{ne}_1\circ\mathsf{ne}_2|\!)}$ is an $(\mathcal{S},FN(\mathsf{ne}_1\circ\mathsf{ne}_2))$-automaton recognising $L(\mathsf{ne}_1\circ\mathsf{ne}_2)$.*

Iteration ne^* : Given a nominal regular expression ne and an $(\mathcal{S},FN(\mathsf{ne}))$-automaton $\mathcal{H}_{(\!|\mathsf{ne}|\!)} = \langle Q, q_0, F, tr\rangle$ which recognises $L(\mathsf{ne})$, the iteration $\mathcal{H}_{(\!|\mathsf{ne}^*|\!)}$ is defined by a tuple $\langle Q^*, q_0^*, F^*, tr^*\rangle$:

- $Q^* \stackrel{\text{def}}{=} Q$;
- $q_0^* \stackrel{\text{def}}{=} q_0$;
- $F^* \stackrel{\text{def}}{=} \{q_0\}$;
- $tr^*(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q,\alpha), & \text{if } q \in Q\backslash F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \\ tr(q,\alpha) \cup \{q_0^*\}, \text{if} & \text{if } q \in F \text{ and } \alpha = \varepsilon \\ tr(q,\alpha), & \text{if } q \in F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \setminus \{\varepsilon\} \end{cases}$

Notice that, since the possible inputs on q in $\mathcal{H}_{(\!|\mathsf{ne}^*|\!)}$ and $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ are the same, here we do not need to consider the "otherwise" case.

Lemma 4. *$\mathcal{H}_{(\!|\mathsf{ne}^*|\!)}$ is an $(\mathcal{S},FN(\mathsf{ne}^*))$-automaton recognising $L(\mathsf{ne}^*)$.*

Name-abstraction $\langle n.\mathsf{ne}\rangle$: Given a nominal regular expression ne and an $(\mathcal{S},FN(\mathsf{ne}))$-automaton $\mathcal{H}_{(\!|\mathsf{ne}|\!)} = \langle Q, q_0, F, tr\rangle$ which recognises $L(\mathsf{ne})$, the name-abstraction $\mathcal{H}_{(\!|\langle n.\mathsf{ne}\rangle|\!)}$ is defined by a tuple $\langle Q^\diamond, q_0^\diamond, F^\diamond, tr^\diamond\rangle$:

- q_s and q_t are new states with $\|q_s\| = 0$ and $\|q_t\| = 0$;
- $Q^\diamond \stackrel{\text{def}}{=} Q \uplus \{q_s, q_t\}$ where we increase $\|q\|$ by 1 for each state $q \in Q$ (hence q_s and q_t are the only states on the 0th layer);
- $q_0^\diamond \stackrel{\text{def}}{=} q_s$;
- $F^\diamond \stackrel{\text{def}}{=} \{q_t\}$;

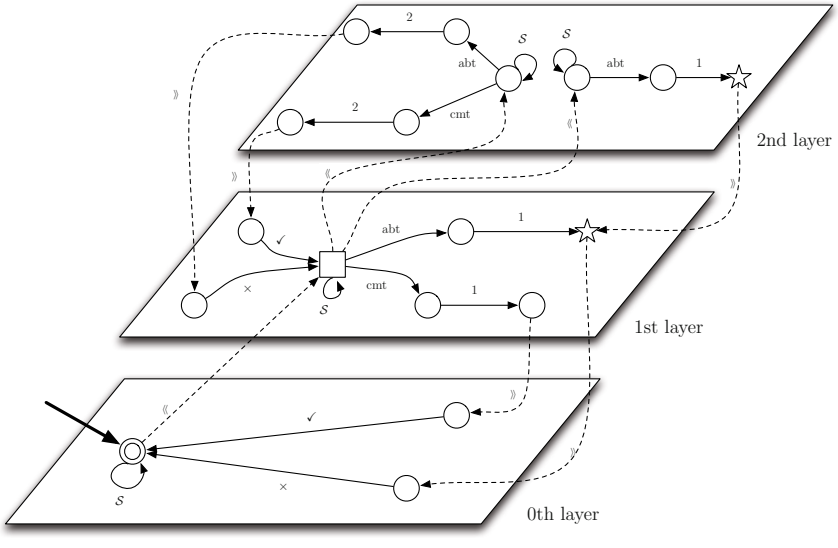
$$- \; tr^{\diamond}(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q,n), & \text{if } q \in Q \text{ and } \alpha = 1 \in \underline{\|q\|} \\ tr(q,\alpha-1), & \text{if } q \in Q \text{ and } \alpha \in \underline{\|q\|} \setminus \{1\} \\ tr(q,\alpha), & \text{if } q \in Q \setminus F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \setminus \underline{\|q\|} \\ \{q_t\}, & \text{if } q \in F \text{ and } \alpha = \text{»} \\ tr(q,\alpha), & \text{if } q \in F \text{ and } \alpha \in I^{\mathcal{N}_{fin}}(q) \setminus \left(\underline{\|q\|} \cup \{\text{»}\}\right) \\ \{q_0\}, & \text{if } q = q_s \text{ and } \alpha = \text{«} \\ \emptyset, & \text{otherwise} \end{cases}$$

Note that on an $(\mathcal{S}, FN(\langle n.\mathsf{ne}\rangle))$-automaton, n cannot be an input on any state. Intuitively, to bind the free name n, we first increase all numbers in all states in $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ (accordingly for the labels on transitions) by 1, and then we allocate the new number 1 in each state in $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ for the new local name obtained by binding n and rename all labels n on each transition in $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ (if they exist) with the number 1.

Lemma 5. *$\mathcal{H}_{(\!|\langle n.\mathsf{ne}\rangle|\!)}$ is an $(\mathcal{S}, FN(\langle n.\mathsf{ne}\rangle))$-automaton recognising $L(\langle n.\mathsf{ne}\rangle)$.*

Lemmas 1, 2, 3, 4 and 5 complete the proof of Proposition 3.

Example 4. We give an account of how the nominal regular expression ne_2 *in Example 1 is translated to an $(\mathcal{S}_{tx}, \emptyset)$-automaton. Below is a simplified graphical representation of the automaton $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)}$.*



(where "box" and "star" shaped states are used to ease the textual description below). This automaton is basically obtained by the above inductive construction but for the following simplifications:

1. *we removed all* ε*-transitions in the obvious way;*
2. *three equivalent states of $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)}$ accessible from the initial state with «-transitions are now unified as the single box state on the* 1*st layer;*

3. *the star state on the* 2*nd layer which had a* $\rangle$*-transition to a distinct state on the* 1*st layer in* $\mathcal{H}_{\langle\!|\mathsf{ne}_2|\!\rangle}$ *is now connected to the star state on the* 1*st layer. Accordingly, non-accessible states are deleted.*

5 Closure Properties

Here we shall discuss the closure properties of nominal languages summarised in

Theorem 3. *Nominal languages are closed under union, intersection, and resource sensitive complementation.*

The notion of resource-sensitive complementation is given in Definition 4 below. Closure under unions is immediate and the construction is the same as the classical one. Similarly, closure under intersections is shown by taking a product of the respective automata; the only difference wrt the analogous construction in the classical theory is that we must take the product layer-wise (otherwise there is no meaningful way to define the values of $\|\_\|$). It remains to discuss complementation.

In our nominal languages, brackets $\langle$ and $\rangle$ are explicitly expressed as syntax. So, for example, $\langle n.w\rangle$ is different from $\langle m.\langle n.w\rangle\rangle$, even when m does not freely occur in $\langle n.w\rangle$. This is important for complementing nominal regular languages since every word has a maximum *depth* of nested binders determined by the regular expression. Define $\partial(\mathsf{ne})$, the *depth of a nominal regular expression* ne as

$$\begin{aligned}
\mathsf{ne} \in \{\varepsilon, 1, 0\} \cup \mathcal{N} \cup \mathcal{S} &\implies \partial(\mathsf{ne}) = 0\\
\mathsf{ne} = \mathsf{ne}_1 + \mathsf{ne}_2 \text{ or } \mathsf{ne}_1 \circ \mathsf{ne}_2 &\implies \partial(\mathsf{ne}) = max(\partial(\mathsf{ne}_1), \partial(\mathsf{ne}_2))\\
\mathsf{ne} = \langle n.\mathsf{ne}\rangle &\implies 1 + \partial(\mathsf{ne})\\
\mathsf{ne} = \mathsf{ne}^* &\implies \partial(\mathsf{ne})
\end{aligned}$$

For example, if $\mathsf{ne} = \langle n.\langle m.s\ m\ n\rangle^* n\ n \circ s\rangle \circ \langle l.s\ l\rangle \circ s$ then $\partial(\mathsf{ne}) = 2$, hence no word in $L(\mathsf{ne})$ can have more than 2 nested binders; therefore the complement of $L(\mathsf{ne})$ has to include words which have finite but unbounded depth, e.g. $\langle n_1.\langle n_2.\cdots\langle n_k.n_1\cdots n_k\rangle\cdots\rangle\rangle$ for any natural number $k > \partial(\mathsf{ne})$. But, it is impossible to accept all these words on any finite $(\mathcal{S}, \mathcal{N}_{fin})$-automaton. Therefore, nominal regular languages are not closed under the standard complementation.

On the other hand, when contemplating nominal languages, is the notion of the standard complementation suitable? We claim that there are two distinct conditions for rejecting words on $(\mathcal{S}, \mathcal{N}_{fin})$-automata:

1. The word is consumed and the automaton finishes in a non-accepting state.
2. The automaton is in a configuration whose word is of the form $\langle n.w\rangle$ and its state is in the highest layer.

The first is the usual non-acceptance condition, while the second one, which we call *overflow* condition, is necessary for $(\mathcal{S}, \mathcal{N}_{fin})$-automata. Informally, the distinction of the two conditions of rejection above can be rephrased as follows:

> Rejection by non-acceptance takes place when the word represents a 'wrong behaviour'; instead, rejection by overflow happens when we do not have enough resources to process the word.

The considerations above lead to the notion of *resource-sensitive complementation*:

Definition 4. *Let* ne *be a nominal regular expression.* The resource sensitive complementation of $L(\mathsf{ne})$ *is the set* $\{w \notin L(\mathsf{ne}) \mid \partial(w) \leq \partial(\mathsf{ne})\}$ *(where the depth of a word is defined as the depth the corresponding expression).*

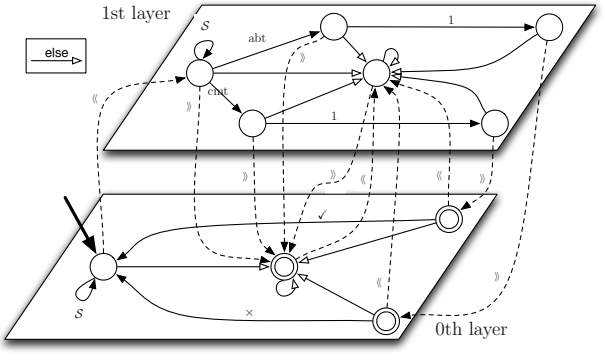
The algebraic structure of union, intersection, and resource sensitive complementation is that of a generalised Boolean algebra [25], that is, of a distributive lattice with bottom and relative complement (but no top).

For the proof that nominal regular expressions are closed under resource-sensitive complementation, note first that the overflow condition characterises the configurations where a deterministic automaton (Definition 1) can get stuck. Further, recall that, by Proposition 3, for each nominal regular expression ne, there is an $(\mathcal{S}, \mathcal{N}_{fin})$-automaton $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ which accepts the language $L(\mathsf{ne})$. By Theorem 1 we can assume, without loss of generality, that $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ is deterministic. Since configurations can get stuck only by overflow, for any word in $L(\mathsf{ne})$, $\mathcal{H}_{(\!|\mathsf{ne}|\!)}$ has a run to a final state which, by construction, is on the 0-th layer. Hence, if we swap the final states with the non-final states on the 0-th layer, the automaton recognises the resource-sensitive complementation of $L(\mathsf{ne})$. Finally, by Proposition 2, we obtain

Proposition 4. *Nominal regular expressions are closed under resource-sensitive complementation.*

We give an example of how to construct an $(\mathcal{S}, \mathcal{N}_{fin})$-automaton which accepts the resource-sensitive complementation of a nominal regular expression in Example 1.

Example 5. An automaton which recognises the resource-sensitive complementation of $L(\mathsf{ne}_1)$ *from Example 1 is given below:*



In this automaton, the transitions with non-filled heads represent "complement transitions", that is transitions taken when the source state does not have any filled-head transition with a label matching the input symbol. Let $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)}$ *be the* $(\mathcal{S}, \mathcal{N}_{fin})$*-automaton in Example 2; the automaton above is constructed from* $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)}$ *as follows:*

1. *firstly* $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)}$ *is determinised, by means of the layer-wise powerset construction; note that this adds deadlock states (in the automaton in the picture above, we just added two deadlock states, one per layer);*

2. *secondly, all non-accessible states are removed;*
3. *finally, accepting and non-accepting states on the* 0*th layer are swapped.*

The automaton in Example 5 can be used as a *transactional monitor* of the transactions characterised by ne_2 in Example 1. Namely, it accepts words representing computations of 2-level nested transactions that diverge from the expected behaviour, e.g., transactions that starts but do not explicitly commit or abort.

6 Conclusion

Our long-term aim is to develop a theory of nominal languages with binders. In this paper we looked at the most basic case where the binders do not interact with the monoid operations. But there is a range of other interesting possibilities. For example, one may impose the additional equation $(\!|n.w|\!) \circ v = (\!|n.w \circ v|\!)$ for n not free in v. This is known as scope extrusion in the π-calculus and would have as a consequence that an automaton recognising $\langle n.n \rangle^*$ would need to be able to keep track of an *unbounded* number of local names (for an analysis of the interplay between binders and name locality see e.g. [20,10]). A first sketch of some of the arising landscape is drafted in [17].

Although the use of binders in this paper is rather restricted, it is expressive enough to represent interesting computational phenomena and it guarantees the properties in § 4 and 5. Increasing the expressiveness of our regular expressions by adding permutations is the natural step we are currently investigating. For instance, we are considering languages where the Kleene-star operator interplays with name automorphisms.

It will be interesting to explore the connections with tree-walking pebble automata [9]. The idea is that the configuration of the automaton is given by a pair (q, v) where q is a state of the automaton and v a node of the input tree. A run is obtained by letting the automaton to change its state and mode to parent/children nodes of v according to the label of v and the fact that v is the root, a leaf, or an internal node. Our approach is reminiscent of this pushdown mechanism of tree-walking pebble automata, with the difference that the decision of dropping/lifting pebbles is driven by binders in the word.

With an eye on applications to verification it is of interest to pursue further developments in the direction of the work [1] on Kleene algebra with local scope. Another direction for future work starts from the observation that, due to the last two clauses of Definition 2, automata with binders do not process words buts nested words (with dangling brackets) [2]. This suggests extend the work of [2] to the nominal setting.

Acknowledgements. The authors thank the reviewers for their criticisms and comments which helped to greatly improve the paper.

References

1. Aboul-Hosn, K., Kozen, D.: Local variable scoping and kleene algebra with tests. J. Log. Algebr. Program. 76(1), 3–17 (2008)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3) (2009)

3. Arbib, M.A., Manes, E.G.: Machines in a category: An expository introduction. SIAM Review 16(163-192), 285–302 (1974)
4. Bojanczyk, M.: Data monoids. In: STACS, pp. 105–116 (2011)
5. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: IEEE Symposium on Logic in Computer Science, pp. 355–364 (2011)
6. Ciancia, V.: Nominal Sets, Accessible Functors and Final Coalgebras for Named Sets. PhD thesis, Dipartimento di Informatica, Università di Pisa (2008) forthcoming
7. Ciancia, V., Montanari, U.: Symmetries, local names and dynamic (de)-allocation of names. Inf. Comput. 208(12), 1349–1367 (2010)
8. Ciancia, V., Tuosto, E.: A novel class of automata for languages on infinite alphabets. Technical Report CS-09-003, Leicester (2009)
9. Engelfriet, J., Hoogeboom, H.J.: Tree-walking pebble automata. In: Jewels are Forever, pp. 72–83 (1999)
10. Fernández, M., Gabbay, M.: Nominal rewriting. Inf. Comput. 205(6), 917–965 (2007)
11. Fiore, M.P., Staton, S.: Comparing operational models of name-passing process calculi. Inf. Comput. 204(4), 524–560 (2006)
12. Gabbay, M., Ciancia, V.: Freshness and Name-Restriction in Sets of Traces with Names. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 365–380. Springer, Heidelberg (2011)
13. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: Symbolic on Logics in Comput Science, pp. 214–224 (1999)
14. Gadducci, F., Miculan, M., Montanari, U.: About permutation algebras (pre)sheaves and named sets. Higher-Order and Symbolic Computation 19(2-3), 283–304 (2006)
15. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison Wesley (2000)
16. Kaminski, M., Francez, N.: Finite-memory automata. TCS 134(2), 329–363 (1994)
17. Kurz, A., Suzuki, T., Tuosto, E.: Towards nominal formal languages. CoRR, abs/1102.3174 (2011)
18. Montanari, U., Pistore, M.: π-Calculus, Structured Coalgebras, and Minimal HD-Automata. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 569–578. Springer, Heidelberg (2000)
19. Pistore, M.: History Dependent Automata. PhD thesis, Dip. di Informatica - Pisa (1999)
20. Pitts, A., Stark, I.: Observable Properties of Higher Order Functions that Dynamically Create Local Names, or What's New? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 122–141. Springer, Heidelberg (1993)
21. Pouillard, N., Pottier, F.: A fresh look at programming with names and binders. In: Proceeding of the 15th ACM SIGPLAN International Conference on Functional Programming, pp. 217–228 (2010)
22. Segoufin, L.: Automata and Logics for Words and Trees over an Infinite Alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
23. Shinwell, M., Pitts, A., Gabbay, M.: Freshml: programming with binders made simple. In: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, pp. 263–274 (2003)
24. Stirling, C.: Dependency Tree Automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 92–106. Springer, Heidelberg (2009)
25. Stone, M.H.: Postulates for boolean algebras and generalized boolean algebras. American Journal of Mathematics 57(4), 703–732 (1935)
26. Tzevelekos, N.: Fresh-Register Automata. In: Symposium on Principles of Programming Languages, pp. 295–306. ACM (2011)
27. Weikum, G., Vossen, G.: Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann (2002)