R. Reussner, A. Koziolek, R. Heinrich (Hrsg.): INFORMATIK 2020,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 987

Complexity Analysis of Task Dependencies in an Artificial Hormone System

Eric Hutter[1], Mathias Pacher[1], Uwe Brinkschulte[1]

Abstract: The Artificial Hormone System (AHS) is a self-organizing tool able to allocate tasks in a distributed system. We extend the AHS in this paper by negator hormones to enable conditional task structures and provide a thorough complexity analysis of the resulting system. The analysis shows that the problem to decide if a given task A is instantiated at all respecting the negators is NP-complete.

Keywords: Artificial Hormone System; negators; conditional task execution; complexity analysis

1 Introduction

We describe and analyze the decision problem NEGATOR-SAT occurring when using an *Artificial Hormone System* (AHS) [BP12] in this paper. The AHS is able to allocate tasks on a set of distributed processors without using a central instance and offers a high dependability of the task allocation. While the original AHS assumes a task model of independent tasks, we extend it here by assuming conditional dependencies between the tasks: E.g. a task T_1 can only be executed when another task T_2 is *not* executed. This allows to enable alternative task structures within the AHS.

Our contribution in this paper is twofold: (1) We shortly describe our extension of the AHS including the *negator hormones*. Their purpose is to enable conditional task structures. (2) Conditional task dependencies induced by negators make it hard to determine if a given task A can be instantiated at all. We call this decision problem NEGATOR-SAT and prove its NP-completeness. We end the paper by providing a transformation example of a satisfiable propositional formula to a task set using negators allowing to instantiate task A.

The paper is structured as follows: Section 2 presents the State of the Art in self-organizing systems. Section 3 gives an introduction to the original AHS while section 4 briefly explains our negator implementation. The complexity analysis of NEGATOR-SAT as well as an example are provided in section 5. Section 6 concludes the paper and describes future work.

2 State of the Art

IBM's *Autonomic Computing* initiative [LMD13] introduced so-called *self-x* properties such as self-configuration, self-optimization and self-healing. The MAPE-K loop was

[1] Goethe University, Frankfurt am Main, Germany, {hutter, pacher, brinks}@es.cs.uni-frankfurt.de

doi:10.18420/inf2020_92

established to realize monitoring and analyzing of a system's behavior and to plan and execute actions controlling its behavior according to a knowledge base and user-defined goals. This loop has recently been adopted to establish self-explainable systems by using a MAB-EX loop (monitor, analyze, build, explain), see [Bl19]. The above mentioned self-x properties are also central to systems realized using *Organic Computing* concepts [TSM17]: Here, computer systems and embedded systems are constructed by incorporating concepts inspired by biological systems and their organization principles. This approach allows systems to dynamically adapt to changing operational conditions, realizing self-x properties like self-configuration or self-healing at run-time.

3 The Artificial Hormone System

The AHS' main purpose is to allocate tasks in a distributed system of processors, called *processing elements* or *PEs*. It is completely decentralized and has no single point of failure. In addition, it provides self-x properties such as self-configuration, self-optimization and self-healing and guarantees real-time bounds [BP12].

The AHS uses different kinds of hormones (which are short messages) to allocate the tasks. The main hormone types are eager values, suppressors and accelerators. Eager values indicate the suitability of a PE to take a task. As soon as a PE takes a task it sends suppressors for it. In this way, it tells the other PEs that it has taken the task: This is a life-sign on the one hand and it saturates the hormone balance on the other hand, thus limiting the number of allocated instances of this task. Accelerators are used to locate related tasks (i.e. tasks with communication relations or access to the same sensors or actors) nearby each other.

The core of the AHS is the hormone loop. Each PE iterates the loop, computing the hormone balance for each task. The duration needed by one hormone loop iteration is called a *hormone cycle*. In the *receive stage*, the hormones for each task are received. In the *compute and decision stage*, the suppressors received for a task are subtracted from its local eager value and the accelerators for this task are added. The result is the modified eager value which indicates the PE's current suitability to take this task. This computation is performed for each task. A PE's AHS instance then decides for a *single* task allocation per hormone loop iteration in order to allow the suppressors and accelerators to unfold their effect. In the following *send stage*, the PEs send eager values for all tasks (with the exception of an eager value that is 0) as well as suppressors and accelerators for all tasks they are currently executing. In this paper, we want to express conditional task relationships using the self-organizing AHS. A conditional task relationship means that a task T_1 can only be executed when another task T_2 is *not* executed. The concept is realized by special hormones called *negators* and allows to use alternative task structures in the AHS. This may be useful in a heterogeneous system of PEs. Details on the negators are described in section 4. Figure 2 gives a sketch of the hormone loop (already including the negators).

4 Conception

As mentioned before, our goal was to enhance the AHS by introducing the possibility to model conditional dependencies between tasks. To be precise, we introduced so-called *negator* relationships between tasks as visualized in Figure 1: Here, task T_j negates task T_i, meaning that task T_i cannot be assigned to any PE if T_j is assigned to a PE. Thinking in terms of a directed graph, this relationship can be expressed as the tuple (T_j, T_i).

Fig. 1: Negator relationship between tasks T_j and T_i: If T_j is assigned, T_i must not be assigned

This allows to express task dependencies: Suppose one PE_α can execute a task T realizing some functionality. Multiple other PEs cannot execute T but rather a set of tasks that realize the same functionality as T but with some kind of degradation, e.g. loss of precision. If PE_α is running, T's negator relationships to the other tasks prevent them from being instantiated. If PE_α fails, T is no longer available but the other tasks can be instantiated, keeping the functionality available.

4.1 Implementation

We implemented our concept of negator relationships that realize conditional inter-task dependencies by modifying the AHS' hormone loop as shown in Figure 2 (cf. [Br13] for

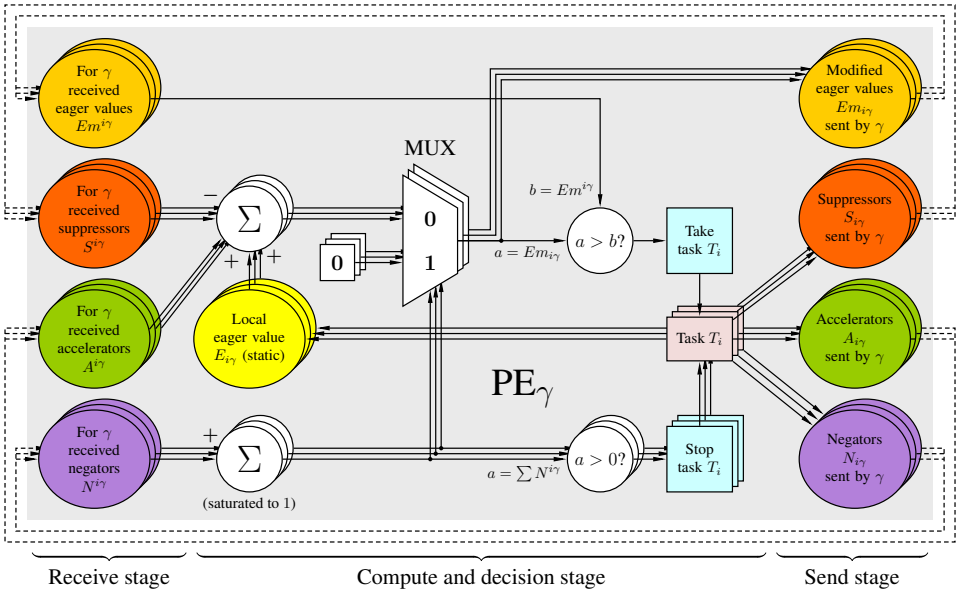


Fig. 2: Hormone loop with negators, running on PE_γ

information on the AHS' original hormone loop): We added an additional type of hormone, the so-called *negators*. If some task T_j is running and a negator relationship (T_j, T_i) exists, T_j will send a negator hormone to T_i during the hormone loop's send stage.

The received negator hormones are counted for each task. If at least one negator was received for some task T_i, two things happen: (1) T_i is stopped if it is running. (2) T_i gets blocked by forcing its modified eager value to 0, regardless of any suppressors or accelerators received for T_i. This prevents T_i from being assigned. If the negating task T_j is no longer assigned to any PE (e.g. because the PE it was running on failed), it won't send a negator for T_i any longer. This allows T_i's modified eager value to rise above 0 again, allowing T_i to be assigned again.

5 Theoretical Analysis

As has been seen in the previous section, the introduction of negators allows to model task dependencies, e.g. alternate sets of tasks to realize some functionality. However, negators also introduce new kinds of possible mistakes a designer can make during a system's design. Consider the following problem:

Definition 1 (NEGATOR-SAT)**.** Let $\mathcal{T}$ be a finite set of tasks and $\mathcal{N} \subseteq \mathcal{T} \times \mathcal{T}$ a set of negator relationships among those tasks.

The decision problem NEGATOR-SAT is now stated as follows: Given a task $A \in \mathcal{T}$, does a set of assigned tasks $\mathcal{V} \subseteq \mathcal{T}$ exist (with $T \in \mathcal{V}$ iff T is assigned to a PE) so that the following conditions are all satisfied:

(1) There is no task $T \in (\mathcal{T} \setminus \mathcal{V})$ that could be assigned to a PE even if all PEs had infinite computational resources,
(2) $\mathcal{V}$ is a stable task assignment, i.e. all negator relationships among tasks from $\mathcal{V}$ are respected,
(3) $A \in \mathcal{V}$, i.e. task A is assigned to some PE.

In simple terms, NEGATOR-SAT asks whether a stable task assignment exists so that A can be assigned to a PE. Condition (1) prevents the system's computational capacities from imposing any limits on such task assignment. Clearly, it can be regarded a design mistake if some task cannot be assigned to a PE at all. Thus, it should be checked if each task is assignable. However, it turns out that this seemingly simple problem is difficult to solve algorithmically:

Theorem 2. NEGATOR-SAT is NP-hard.

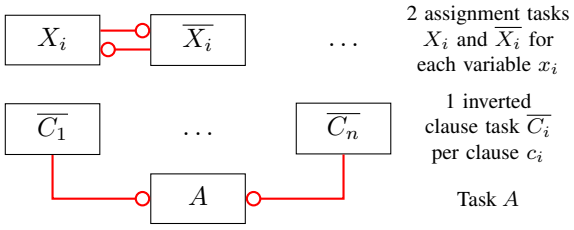
Proof. By reduction of 3-SAT to NEGATOR-SAT: We will show 3-SAT $\leq_p$ NEGATOR-SAT where $\leq_p$ denotes a polynomial-time reduction. If 3-SAT is reducible to NEGATOR-SAT in polynomial time, we can deduce that NEGATOR-SAT is at least as hard as 3-SAT. With 3-SAT being NP-complete, the NP-hardness of NEGATOR-SAT follows.

We will thus describe a transformation τ so that

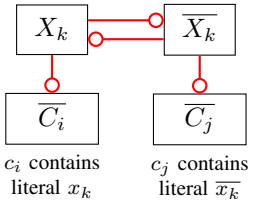
(i) τ can be computed in polynomial time w.r.t. the input length and
(ii) $f \in$ 3-SAT $\iff \tau(f) \in$ NEGATOR-SAT.

Let $f := \bigwedge_{i=1}^{n} c_i$ with $c_i := (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ be a 3-SAT formula with $l_{i,j} \in \{x_k, \overline{x_k}\}$ for some k.[2] We will transform f into a task set $\mathcal{T}$ with negator relationships $\mathcal{N}$ so that the transformed input $\tau(f) := (\mathcal{T}, \mathcal{N}, A)$ is an instance of NEGATOR-SAT for the task $A \in \mathcal{T}$.

Construction of τ. The basic construction principle of this transformation is shown in Figure 3a. For each variable x_k occurring in f, we create two *assignment tasks* X_k and $\overline{X_k}$ that negate each other. This ensures only one of them can be assigned in a stable system, representing x_k's interpretation. Condition (1) ensures at least one of those assignment tasks is assigned per variable while condition (2) ensures that both cannot be assigned simultaneously.



(a) Basic construction principle



(b) Negator relationships between assignment and inverted clause tasks

Fig. 3: Construction principle of transformation from 3-SAT to NEGATOR-SAT

Additionally, we introduce one *inverted clause task* $\overline{C_i}$ per clause c_i. Thus, the resulting task set is

$$\mathcal{T} = \bigcup_{x_k \in \text{Variables}(f)} \{X_k, \overline{X_k}\} \cup \bigcup_{i=1}^{n} \{\overline{C_i}\} \cup \{A\}$$

where n is the number of clauses in f.

We will ensure that each inverted clause task $\overline{C_i}$ can (and per condition (1) will) be assigned iff f's interpretation does *not* satisfy the corresponding clause c_i by introducing negator relationships as follows (cf. Figure 3b):

- $(X_k, \overline{C_i}) \in \mathcal{N} \iff c_i$ contains the literal x_k and
- $(\overline{X_k}, \overline{C_i}) \in \mathcal{N} \iff c_i$ contains the literal $\overline{x_k}$.

Finally, the following negator relationships ensure that task A can (and, again, per condition (1) will) be assigned iff no inverted clause task $\overline{C_i}$ is assigned (and thus, all corresponding clauses c_i are satisfied): $(\overline{C_i}, A) \in \mathcal{N}$ for all $1 \le i \le n$.

[2] For simplicity, we require that each clause in f consists of exactly three literals. Note that this restriction does not change the problem's complexity as a clause can be padded to exactly three literals by repeating one of its literals.

See section 5.1 for an example of this construction.

τ is a polynomial-time reduction. We now need to show that above claims (i) and (ii) hold for τ, i.e. that τ is indeed a reduction of 3-SAT to NEGATOR-SAT.

(i): Polynomial time: It is easy to see that a formula f with n clauses and v different variables (with $v \leq 3n$) can be transformed in polynomial time w.r.t. f's length: We only need to construct $2v$ assignment tasks, n inverted clause tasks and the task A. This sums to $2v + n + 1 \leq 7n + 1$ tasks which is polynomial in the input formula's length.

Additionally, we construct $2v$ negator relationships for mutual exclusion of X_k and $\overline{X_k}$, $3n$ negator relationships between X_k resp. $\overline{X_k}$ and $\overline{C_i}$ and n negator relationships between $\overline{C_i}$ and A. This totals at $2v + 4n \leq 10n$ relationships which is also polynomial in the input formula's length.

(ii), part 1: $f \in$ 3-SAT $\Rightarrow \tau(f) \in$ NEGATOR-SAT:

Proof. Since $f \in$ 3-SAT, there must exist a satisfying interpretation $\mathcal{I}$: Variables$(f) \to \{0, 1\}$. Thus, consider the set $\mathcal{V} \subseteq \mathcal{T}$ of assigned tasks given as follows:

- $A \in \mathcal{V}$,
- for each inverted clause task $\overline{C_i}$: $\overline{C_i} \notin \mathcal{V}$,
- $X_k \in \mathcal{V} \iff \mathcal{I}(x_k) = 1$ and $\overline{X_k} \in \mathcal{V} \iff \mathcal{I}(\overline{x_k}) = 1$.

Due to τ's construction, it is easy to see that $\mathcal{V}$ satisfies conditions (1) to (3) as given by Definition 1:[3]

(1) All tasks in $(\mathcal{T} \setminus \mathcal{V})$ have an inbound negator link coming from an assigned task, thus no additional task can be instantiated.
(2) All negator relationships are respected: No two tasks from $\mathcal{V}$ share a negator relationship.
(3) A is assigned. ◇

(ii), part 2: $\tau(f) \in$ NEGATOR-SAT $\Rightarrow f \in$ 3-SAT:

Proof. Let $\mathcal{V} \subseteq \mathcal{T}$ be a set of assigned tasks so that conditions (1) to (3) as given by Definition 1 are satisfied. Thus, task A must be assigned. Therefore, per condition (2), no inverted clause task $\overline{C_i}$ can be assigned. Thus, at least one assignment task per inverted clause task $\overline{C_i}$ must be assigned (else, $\overline{C_i}$ would have to be assigned per condition (1)). Additionally, per condition (2), for each assignment task X_k resp. $\overline{X_k}$, the inverse assignment task $\overline{X_k}$ resp. X_k cannot be assigned.

This allows to construct an interpretation $\mathcal{I}$ for f so that $\mathcal{I}(x_k) = 1 \iff X_k \in \mathcal{V}$ and

[3] Note that—since $\mathcal{I}$ satisfies f—at least one literal is satisfied for each clause c_i, thus the corresponding inverted clause task $\overline{C_i}$ is not assigned. Since all inverted clause tasks are *not* assigned, A can (and per condition (1) will) be assigned to some PE.

$\mathcal{I}(\overline{x_k}) = 0 \iff \overline{X_k} \in \mathcal{V}$. In addition, $\mathcal{I}$ must satisfy f: Suppose $\mathcal{I}$ would not satisfy f. Then, there must be a clause c_i in f so that $\mathcal{I}$ does not satisfy any of its literals $l_{i,j}$. Due to τ's construction, this would mean that the inverse clause task $\overline{C_i}$ must be assigned per condition (1) which forbids A's assignment per condition (2). ⋄

Final remarks. Since τ can be constructed in polynomial time w.r.t. the input length, it follows that τ is indeed a polynomial-time reduction from 3-SAT to NEGATOR-SAT. Thus, NEGATOR-SAT is at least as hard as 3-SAT. Since 3-SAT is NP-complete, the NP-hardness of NEGATOR-SAT follows. □

However, NEGATOR-SAT is not only NP-hard, but also complete for NP:

Theorem 3. NEGATOR-SAT is NP-complete.

Proof. Let $(\mathcal{T}, \mathcal{N}, A)$ be an input consisting of a set of task $\mathcal{T}$, a set of negator relationships $\mathcal{N}$ and a task $A \in \mathcal{T}$. A nondeterministic Turing machine can now nondeterministically select a subset $\mathcal{V} \subseteq \mathcal{T}$ of assigned tasks and then deterministically check that conditions (1) to (3) from Definition 1 are all satisfied:

(1) This condition is satisfied if, for each $T \in (\mathcal{T} \setminus \mathcal{V})$, there is a negator relationship $(T', T) \in \mathcal{N}$ so that $T' \in \mathcal{V}$. However, this can be checked in $O(\text{poly}(|\mathcal{T}|, |\mathcal{N}|))$ time.
(2) This condition is satisfied if, for each $T \in \mathcal{V}$, there is *no* negator relationship $(T', T) \in \mathcal{N}$ so that $T' \in \mathcal{V}$. This can also be checked in $O(\text{poly}(|\mathcal{T}|, |\mathcal{N}|))$ time.
(3) This condition is satisfied if $A \in \mathcal{V}$ which can be checked in $O(\text{poly}(|\mathcal{V}|))$ time.

The Turing machine shall accept the input iff all three conditions are satisfied.

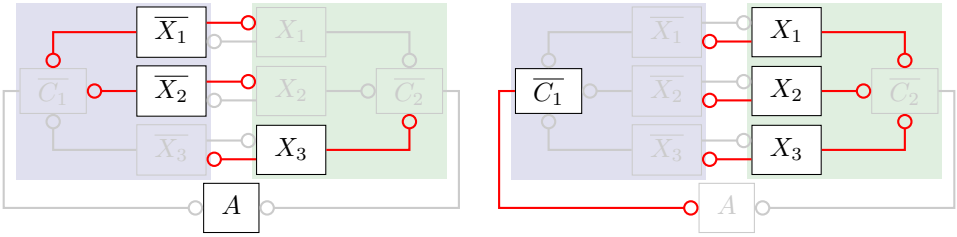
Since, after nondeterministically guessing $\mathcal{V}$, the verification can be performed in polynomial time w.r.t. the input length, it follows that NEGATOR-SAT ∈ NP. Together with Theorem 2, it follows that NEGATOR-SAT is NP-complete. □

This shows the power introduced by negators: Unless P = NP holds, it is not possible to decide in deterministic polynomial time whether a given task A can be assigned to a PE at all (when requiring a stable task assignment in which no additional tasks can be assigned).

5.1 Example of Construction

Figure 4 shows the construction result $\tau(f)$ for the formula $f = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$. Note that f is satisfiable and hence $f \in$ 3-SAT. It is easy to see that assigning either X_i or $\overline{X_i}$ for $i \in \{1, 2, 3\}$ allows assignment of A iff the assignment corresponds to a satisfying interpretation of f.

Additional examples of this construction and further considerations on problems involving negators can be found in [HPB20].



(a) With task assignment corresponding to satisfying interpretation of f

(b) With task assignment corresponding to unsatisfying interpretation of f

Fig. 4: NEGATOR-SAT instance constructed from 3-SAT instance $f = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$

6 Conclusion

We presented an negator extension for the AHS middleware in this paper. Negators enable conditional task execution in the AHS which is useful in a heterogeneous processor system. The use of negators impose the problem to determine if a given task A can be instantiated in the context of a stable task allocation in the overall system. We called the problem NEGATOR-SAT and proved its NP-completeness. Future work will consider the negators' impact on the AHS' real-time bounds and the stability of task allocations. This is important as it is simple to see that negators can generate oscillating task allocations.

Bibliography

[Bl19] Blumreiter, Mathias; Greenyer, Joel; Garcia, Francisco Javier Chiyah; Klös, Verena; Schwammberger, Maike; Sommer, Christoph; Vogelsang, Andreas; Wortmann, Andreas: Towards Self-Explainable Cyber-Physical Systems. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019. pp. 543–548, 2019.

[BP12] Brinkschulte, Uwe; Pacher, Mathias: An Agressive Strategy for an Artificial Hormone System to Minimize the Task Allocation Time. In: 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, ISORC Workshops 2012, Shenzhen, China, April 11, 2012. pp. 188–195, 2012.

[Br13] Brinkschulte, Uwe; Pacher, Mathias; von Renteln, Alexander; Betting, Benjamin: Organic Real-Time Middleware. In (Higuera-Toledano, M. Teresa; Brinkschulte, Uwe; Rettberg, Achim, eds): Self-Organization in Embedded Real-Time Systems, pp. 179–208. Springer New York, New York, NY, 2013.

[HPB20] Hutter, Eric; Pacher, Mathias; Brinkschulte, Uwe: On the Hardness of Problems Involving Negator Relationships in an Artificial Hormone System. arXiv:2006.08958 [cs], June 2020.

[LMD13] Lalanda, Philippe; McCann, Julie A.; Diaconescu, Ada: Autonomic Computing - Principles, Design and Implementation. Undergraduate Topics in Computer Science. Springer, 2013.

[TSM17] Tomforde, Sven; Sick, Bernhard; Müller-Schloer, Christian: Organic Computing in the Spotlight. arXiv:1701.08125 [cs], January 2017.