Computation on Structures
Behavioural Theory, Logic, Complexity

Klaus-Dieter Schewe(✉)

UIUC Institute, Zhejiang University, Haining, China
kd.schewe@intl.zju.edu.cn

Abstract. Over the last decades the field of computer science has changed a lot. In practice we are now dealing with very complex systems, but it seems that the theoretical foundations have not caught up with the development. This article is dedicated to a demonstration how a modernised theory of computation may look like. The theory is centred around the notion of algorithmic systems addressing behavioural theory, logic and complexity theory.

Keywords: Theory of computation · Behavioural theory · Computation on structures · Complexity theory · Logic · Abstract State Machines · Parallel algorithms · Insignificant choice · PTIME

Dear Egon,

The first time we met was at a Dagstuhl seminar in 1997 organised by Bernhard Thalheim. You were sceptical concerning my presentation on consistency enforcement in formal specifications due to the use of predicate transformers, but I think I could convince you that the existence proof (in infinitary logic) is possible, though the doubts concerning their usefulness remained. We also discussed about your 1985 monograph on computation theory, logic and complexity in the field of computer science, which in the very same year I used as a text in an introductory course on Theoretical Computer Science. Though some of the material was considered very demanding for the students, it was (and still is) one of the best texts describing the links between computation theory, logic and complexity theory, as it was handled until that time.

It took years to meet again, because Egon started to develop the very successful use of Abstract State Machines (ASMs) for rigorous software development, while I had turned my back on "formal methods" after discovering how little the FM community was interested in mathematical foundations. This changed again after getting to know ASMs better. I tentatively started putting Ph.D. students on the track, and in one case the intended quick exploitation of ASMs for database transformations became the basis of a convincing theory of parallel algorithms.

More than 35 years passed since the publication of your monograph on computation theory, logic and complexity, and over this period the field of computer

A. Raschke et al. (Eds.): Börger Festschrift, LNCS 12750, pp. 266–282, 2021.
https://doi.org/10.1007/978-3-030-76020-5_15

science has changed a lot. In practice we are now dealing with complex systems of systems, but though ASMs turned out very suitable to cover the developments, it seems that the theoretical foundations have not caught up with it. This birthday present is dedicated to a demonstration how a modernised theory of computation may look like. The theory is centred around the notion of algorithmic systems, which are harder to define than computable functions, in particular, when all developments in computing are to be taken into account.

I will argue that behavioural theories are key to the understanding, i.e. we require language-independent axiomatic definitions of classes of algorithmic systems that are accompanied by abstract machine models provably capturing the class under consideration. The machine models give further rise to tailored logics through which properties of systems in the considered class can be formalised and verified, and to fine-tuned classifications on the grounds of complexity restrictions. I will outline that all extensions will be (1) *conservative* in the sense that the classical theory of computation is preserved, (2) *universal* in the sense that all practical developments are captured uniformly, and (3) *practical* in the sense that languages associated with the abstract machine models can be used for rigorous high-level systems design and development, and the logics can be exploited for rigorous verification of desirable properties of systems. This links to your newer monographs focusing on the theory and application of the Abstract State Machine method.

1 Towards a Theory of Computation on Structures

In 1985 Egon Börger published his influential monograph on computation theory, logic and complexity (see the English translation in [8]), which focused on the concept of *formal language* as carrier of the precise expression of meaning, facts and problems, and the concept of *algorithm* or calculus, i.e. a formally operating procedure for the solution of precisely described questions and problems. At that time the text was at the forefront of a modern theory of these concepts, paving the way in which they developed first in mathematical logic and computability theory and later in automata theory, theory of formal languages and complexity theory.

Nonetheless, it became clear that the state of the theory left many open problems. Computing started to stretch out into many new application areas. Distributed computing over networks became possible, database systems facilitated concurrent computation, artificial intelligence ventured from a niche area to a useful technology enabling inferential problem solving in diagnosis, controlling machines through software became possible, etc. Now, only 35 years later the rapid progress in computing has led to a fascinating variety of interconnected systems that are used to support, manage and control many aspects of our life. There is hardly an area that has not yet been penetrated by computing, and still there are many open challenges for the continuation of this success story.

We are now dealing with systems of systems that are

- operating in *parallel* exploiting synchronously multiple processor cores and asynchronously computing resources distributed over networks,
- *hybrid* interacting with analogue systems with continuous behaviour,
- *adaptive* changing their own behaviour,
- *intelligent* reasoning about themselves and their environment,
- *interactive* communicating with their environment, and
- *random* depending on probability distributions.

All these developments require scientific foundations centred around computation theory, complexity and logic:

- Is there a theory of computation that faithfully covers all the aspects of systems of computing systems that occur in practice?
- Is there a methodology grounded in such a theory of computation that permits the definition and classification of complex systems and the provision of means for specification, systematic development, validation and verification?
- Is there a methodology that permits reasoning about problems and their solutions in terms of correctness and complexity?

In 1982 Chandra and Harel raised the problem, whether there exists a computation model over structures that captures the complexity class PTIME rather than Turing machines that operate over finite strings [17]. The problem reflects the typically huge gap between the abstraction level of an algorithm or more generally a system of algorithmic systems and the level of Turing machines. It is not sufficient to know that deep inside the core of systems we deal with computations that given a proper string encoding can be represented by Turing machines; instead, computation theory has to stretch to arbitrary Tarski structures that are omnipresent in all mathematical theories, and any extension should be conservative in the sense that the classical theory is preserved as a representation on the lowest level of abstraction.

A first answer was given in 1985 by Gurevich's "new thesis" [26], which was further elaborated in the 1995 Lipari guide [28]. The new theory emphasises Tarski structures (aka universal algebras) to capture abstract states of systems and evolving algebras, now known as Abstract State Machines (ASMs), as the abstract machines capturing the algorithms on arbitrary levels of abstraction. Egon Börger realised that these ideas do not only create a new paradigm for the foundations of computing subsuming the classical theory, but at the same can be exploited for rigorous systems engineering in practice thereby fulfilling the criteria of a "software engineering" discipline that deserves this name as envisioned in the 1968 meeting in Garmisch, where this notion was coined [32].

A remarkable success story started leading to proofs of compiler correctness for the Warren Abstract Machine for Prolog [13], the translation from Occam to transputers [10], the compilation of Java and the bytecode verifier [37], the development of the sophisticated theory of ASM refinements [9], and much more. The state of the theory and practice of ASMs is well summarised in Egon Börger's

and Robert Stärk's monograph on ASMs [16]. More recent examples are found in the modelling companion by Börger and Raschke [12].

While the development proved that ASMs can take over the role of the formal languages in computation theory, it took until 2000 to develop the celebrated "sequential ASM thesis" [29], which is based on the observation that "if an abstraction level is fixed (disregarding low-level details and a possible higher-level picture) and the states of an algorithm reflect all the relevant information, then a particular small instruction set suffices to model any algorithm, never mind how abstract, by a generalised machine very closely and faithfully". On one hand the thesis provided a language-independent definition of the notion of *sequential algorithm* giving for the first time in history a precise axiomatic definition of the notion of "algorithm" (though restricted to sequential algorithms). On the other hand it contained the proof that all algorithms as stipulated by the defining postulates are faithfully captured by sequential ASMs. This justified further to establish another new notion: a *behavioural theory* comprises a machine-independent axiomatic definition of a class of algorithms (or more generally: algorithmic systems), an abstract machine model, and a proof that the machine model captures the class of computations.

Starting from the first behavioural theory, the theory of sequential algorithms, further success stories followed. Moschovakis's critical question how recursion could be captured was answered by the behavioural theory of recursive algorithms [15]. A first attempt to extend the theory to parallel algorithms was undertaken by Blass and Gurevich [5], but it was not well received due to the use of concepts such as mailbox, display and ken that were considered too close to the machine model, but another behavioural theory of parallel algorithms without these restrictions was then developed in [22]. This closed the case of synchronous parallel algorithms. A convincing behavioural theory for asynchronous algorithmic systems was developed in [14] with *concurrent ASMs* as the machine model capturing concurrent algorithms, i.e. families of sequential or parallel algorithms associated with agents that are oblivious to the actions of each other apart from recognising changes to shared locations. Recently, a behavioural theory of reflective algorithms was developed addressing the question how to capture algorithmic systems that can adapt their own behaviour [34].

The behavioural theories yield variants of Abstract State Machines that can be used for rigorous systems development. Furthermore, Stärk and Nanchen developed a logic for the reasoning about deterministic ASMs [36]. As discussed in [16] it was considered difficult to extend this logic to the case of non-deterministic ASMs[1]. This gap was closed in [23] by making update sets first-class objects in the theory and proving completeness with respect to Henkin semantics. It was also shown how the logic can be adapted to reason about concurrent ASMs [24]. An extension to reflective ASMs was approached in [35]. On one side it shows the tight connections between the classes of algorithmic systems handled in the behavioural theories. On the other side it shows that the development of the logical counterpart of the theories has not yet reached the same development state.

[1] Note a full behavioural theory of non-deterministic algorithms does not yet exist.

This applies even more so to complexity theory. One of the few studies trying to bring complexity theory to the theory of ASMs, which after all provide the theory of computations on structures as asked for by Chandra and Harel, is the theory of *choiceless polynomial time* (CPT) [6,7], which studies the choiceless fragment of PTIME using PTIME bounded deterministic Abstract State Machines. Though it was possible to show that CPT subsumes other models of computation on structures[2] such as relational machines [3], reflective relational machines [1] and generic machines [2], it is strictly included in PTIME. If the hope had been to exhaust PTIME the same as existential second-order logic captures NP [21], this failed. No systematic research trying to close the gap between CPT and PTIME followed, and Gurevich posted his conjecture that there is no logic capturing PTIME [27].

If true, it would doom all further attempts in this direction. This would further imply that complexity theory as a whole, in particular descriptive complexity theory [30] which is tighly coupled with finite model theory [20,31], could not be based on more abstract models of computations on structures. In particular, it would not be possible to avoid dealing with string encodings using Turing Machines. However, this consequence appears to be less evident in view of the ASM success stories. Various attempts have been undertaken to refute Gurevich's conjecture either by adding quantifiers such as counting [19] or by adding non-deterministic choice operators [4,25]. A comparison and evaluation is contained in [18].

All these attempts failed, and the main reason for the failure is the neglection of the computations understood as yielding sequences of abstract states with update sets defining the state transitions. Instead, only the functional relationship between the input structure and the Boolean output was emphasised. This restriction to Boolean queries blurs the subtle distinctions that become possible, when the behavioural theory and the associated logic are taken into account. A refutation of Gurevich's conjecture has been achieved in [33] exploiting *insignificant choice*[3] thus leading to *insignificant choice polynomial time* (ICPT). Based on the insight that choice is unavoidable to capture PTIME it is not too hard to see that PTIME problems can be solved by polynomial time bounded ASMs with insignificant choice, as it suffices to create an order on the set of atoms in the base set. This construction is rather specific, as it exploits to choose only atoms, and it permits to replace arbitrary insignificant choice ASMs by ASMs satisfying a *local insignificance condition*. This condition can be expressed in the logic of non-deterministic ASMs [23,24]. To show that the extension remains within PTIME it suffices to simulate PTIME ASMs with choices among atoms that satisfy the local insignificance condition by PTIME Turing machines with input strings given by the standard encoding of an ordered version of the input structure. Here the local insignificance permits to choose always the smallest atom,

[2] Strictly speaking, all these previous computational models are still based on Turing machines, which are coupled with queries on relational stores.

[3] Insignificant choice imposes two conditions on the update sets yielded by a choice. The first of these conditions is similar to semi-determinism [38].

and the PTIME bound results from the fact that local insignificance checking for choices among atoms can be done in polynomial time. With this logic capturing PTIME it then becomes possible to show that PTIME and NP differ [33].

In the remainder of this article I will further elaborate how behavioural theories, associated logics and complexity work together. The emphasis will be on parallel algorithms. In Sect. 2 I will start from the behavioural theory of parallel algorithm, which will be extended by insignificant choice. This does not alter the expressiveness, but justifies the use of choice rules in many practical examples using ASMs [12]. In Sect. 3 I will proceed with the logic of non-deterministic ASMs and outline how it needs to be modified to capture only insignificant choice. Finally, Sect. 4 brings in polynomial time, where the presence or absence of choice makes a significant difference. In fact, it is the difference between CPT and ICPT. I conclude with a brief outlook in Sect. 5 emphasising that this is just a brief demonstration of how a modernised theory of computation centred around the notion of algorithmic systems may look like.

2 Parallel Algorithms

Let us briefly review the parallel ASM thesis [22], and extend the theory by insignificant choice as in [33]. Note that different from classical computation theory the behavioural theory characterises the class of parallel algorithms by four postulates and then proves that the class is captured by the Abstract State Machines, which is more than just defining the semantics of ASMs.

2.1 The Parallel ASM Thesis

Deterministic algorithms proceed in steps, which is reflected in the sequential time postulate for sequential algorithms [29]. Parallel algorithms[4] do not make a change here; only the amount of updates characterising the transition from a state to its successor varies significantly.

Postulate 1 (Sequential Time Postulate). A *parallel algorithm* $\mathcal{A}$ comprises a non-empty set $\mathcal{S}$ of *states*, a non-empty subset $\mathcal{I} \subseteq \mathcal{S}$ of *initial states*, and a one-step transformation function $\tau : \mathcal{S} \to \mathcal{S}$.

Same as for sequential algorithms a *state* has to reflect all the relevant information, so we also preserve the abstract state postulate, which characterises states as Tarski structures over a fixed signature, i.e. a set of function symbols.

Postulate 2 (Abstract State Postulate). Every *state* $S \in \mathcal{S}$ of a parallel algorithm is a structure over a fixed finite signature Σ such that both $\mathcal{S}$ and $\mathcal{I}$ are closed under isomorphisms, the one-step transformation τ of $\mathcal{A}$ does not change the base set of any state, and if two states S and S' are isomorphic via $\zeta : S \to S'$, then $\tau(S)$ and $\tau(S')$ are also isomorphic via ζ.

[4] More precisely: unbounded parallel algorithms, as sequential algorithms algorithms already subsume bounded parallelism. The difference is that in the unbounded case the parallel branches of a computation depend on the state.

These two postulates alone give already rise to several decisive definitions. A *run* of a parallel algorithm $\mathcal{A}$ is a sequence $S_0, S_1, \ldots$ of states with $S_0 \in \mathcal{I}$ and $S_{i+1} = \tau(S_i)$ for all $i \geq 0$. A *location* of state S is a pair $(f, (a_1, \ldots, a_n))$ with a function symbol $f \in \Sigma$ of arity n and an n-tuple of elements a_i of the base set of S. The *value* $val_S(\ell)$ of a location ℓ in state S is $f_S(a_1, \ldots, a_n)$ using the interpretation f_S of f in S. An *update* in S is a pair (ℓ, v) comprising a location ℓ of S and an element v of the base set of S. An *update set* in S is a set of such updates.

An update set Δ is called *consistent* iff $(\ell, v_1), (\ell, v_2) \in \Delta$ imply $v_1 = v_2$. For a consuistent update set Δ in S we obtain a state $S' = S + \Delta$ with $val_{S'}(\ell) = v$ for $(\ell, v) \in \Delta$, and $val_{S'}(\ell) = val_S(\ell)$ otherwise. Any two states S, S' with the same base set define a unique minimal consistent update set $\Delta(S)$ with $S' = S + \Delta(S)$. In particular, we write $\Delta_{\mathcal{A}}(S)$ for the update set defined by S and its successor $\tau(S)$.

Update sets $\Delta_{\mathcal{A}}(S)$ must be determined by the parallel algorithm, which has an intrinsic finite representation. For sequential algorithms it suffices to assume that this finite representation contains a finite set of ground terms over the signature Σ such that the evaluation of these terms in a state S uniquely determines the updates in S. This gives rise to the *bounded exploration postulate*. For parallel algorithms this is slightly more complicated, as in every state the algorithm may execute an arbitrary number of parallel branches. However, these branches are determined by the state. As there must exist a finite representation, it is justified to assume that the branches are determined by terms, so it suffices to replace the ground terms by multiset comprehension terms[5].

Postulate 3 (Bounded Exploration Postulate). Every parallel algorithm $\mathcal{A}$ of signature Σ comprises a finite set W (called *bounded exploration witness*) of multiset comprehension terms $\{\!\{t(\bar{x}, \bar{y}) \mid \varphi(\bar{x}, \bar{y})\}\!\}_{\bar{x}}$ over signature Σ such that $\Delta_{\mathcal{A}}(S) = \Delta_{\mathcal{A}}(S')$ holds, whenever the states S and S' of $\mathcal{A}$ coincide on W.

Finally, each computation has a *background* comprising the implicit fixed values, functions and constructors that are exploited, but not defined in the signature. For sequential algorithms the background was kept implicit, as it merely requires the presence of truth values and the usual operators on them, a value *undef* to capture partial functions, and an infinite reserve, from which new values can be taken if necessary. Parallel algorithms must in addition require the presence of tuples and multisets as already used for bounded exploration. This leads to the *background postulate*.

Postulate 4 (Background Postulate). Each parallel algorithm $\mathcal{A}$ comprises a background class $\mathcal{K}$ defining at least a binary *tuple constructor* and a *multiset constructor* of unbounded arity, and a background signature Σ_B contains at least the following static function symbols:

[5] It must be multiset terms and not set terms, as there may be multiple branches doing the same.

- nullary function symbols `true`, `false`, `undef` and $\{\!\{\}\!\}$,
- unary function symbols `reserve`, `Boole`, $\neg$, `first`, `second`, $\{\!\{\cdot\}\!\}$, $\biguplus$ and `AsSet`, and
- binary function symbols $=$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\uplus$ and $(\,,\,)$.

We assume general familiarity with Abstract State Machines [16], so we will not define them here. Then the key result in [22] is the following "parallel ASM thesis".

Theorem 1. *Abstract State Machines capture parallel algorithms as defined by the sequential time, abstract state, bounded exploration and background postulates.*

The proof that ASMs fulfil the requirement of the Postulates 1–4 is not very difficult. A bounded exploration witness can be constructed from an ASM rule; then showing the decisive property of Postulate 3 is rather straightforward.

The proof that every parallel algorithm as stipulated by the four postulates can be step-by-step simulated by an ASM with the same background and signature is complicated. The key argument is to show that if an update set $\Delta_{\mathcal{A}}(S)$ contains an update $((f,(a_1,\ldots,a_n)),a_0)$, then any $(n+1)$-tuple $(b_0,\ldots,b_n)$ with the same type as $(a_0,\ldots,a_n)$ also defines an update $((f,(b_1,\ldots,b_n)),b_0) \in \Delta_{\mathcal{A}}(S)$, where the *type* is defined by a bounded exploration witness W. Exploting *isolating formulae* for types gives rise to a **forall**-rule r_S with $\Delta_{r_S}(S) = \Delta_{\mathcal{A}}(S)$. The extension to a single rule r with $\Delta_r(S) = \Delta_{\mathcal{A}}(S)$ for all states S uses the same ideas as the proof of the sequential ASM thesis with straightforward generalisations.

2.2 Parallel Algorithms with Choice

As shown by many examples in [12,16] it is often useful to permit non-deterministic choice. We will therefore explore how to extend the parallel ASM thesis to a non-deterministic parallel ASM thesis, then restrict choice such that it becomes insignificant, i.e. the final result does not depend on the choice (up to isomorphism).

Clearly, the abstract state and background postulates can be preserved, but the sequential time postulate has to be replaced by a *branching time postulate*.

Postulate 5 (Branching Time Postulate). A *non-deterministic parallel algorithm* $\mathcal{A}$ comprises a non-empty set $\mathcal{S}$ of *states*, a non-empty subset $\mathcal{I} \subseteq \mathcal{S}$ of *initial states*, and a one-step transformation relation $\tau \subseteq \mathcal{S} \times \mathcal{S}$.

We continue to call each state $S' \in \tau(S)$ a *successor state* of S. Then S and $S' \in \tau(S)$ define a unique minimal consistent update set $\Delta(S,S')$ with $S + \Delta(S,S') = S'$. Let $\boldsymbol{\Delta}_{\mathcal{A}}(S) = \{\Delta(S,S') \mid S' \in \tau(S)\}$ denote the *set of update sets* in state S.

In the same way as the shift from sequential algorithms to parallel algorithms required multiset comprehensions, it seems plausible that also the shift from

parallel algorithms to non-deterministic parallel algorithms will require multiset comprehensions. We therefore define a *witness term* as a term of the form

$$\{\!\{ \{\!\{ t(\bar{x},\bar{y}) \mid \varphi(\bar{x},\bar{y}) \}\!\} \mid \psi(\bar{x}) \}\!\} .$$

Then the bounded exploration postulate could be altered as follows:

Postulate 6 (Non-Deterministic Bounded Exploration Postulate). Every non-deterministic parallel algorithm $\mathcal{A}$ of signature Σ comprises a finite set W (called *bounded exploration witness*) of witness terms over signature Σ such that $\boldsymbol{\Delta}_{\mathcal{A}}(S) = \boldsymbol{\Delta}_{\mathcal{A}}(S')$ holds, whenever the states S and S' of $\mathcal{A}$ coincide on W.

It is again rather straightforward to show that non-deterministic ASMs satisfy the modified postulated for non-deterministic parallel algorithm.

Theorem 2. *Non-deterministic Abstract State Machines define non-deterministic parallel algorithms as defined by the branching time, abstract state, non-deterministic bounded exploration and background postulates.*

However, a proof that non-deterministic Abstract State Machines capture non-deterministic parallel algorithms has not yet been completed. This will be dealt with elsewhere. Our interest here is on a restricted version of non-determinism.

In a run $S_0, S_1, \ldots$ we call a state S_n *final* iff $\tau(S_n) = \{S_n\}$ holds, i.e. there is no more change to the state. Assuming that some function symbols in Σ have been declared as *output functions*. Let $out(S)$ denote the restriction of a final state S to its output locations. Then we call a non-deterministic parallel algorithm $\mathcal{A}$ a *insignificant choice algorithm* iff every run has a final state and for any two final states S_1 and S_2 the outputs $out(S_1)$ and $out(S_2)$ are isomorphic.

Next consider *locally insignificant choice* ASMs, i.e. non-deterministic ASMs with the following properties:

(i) For every state S any two update sets $\Delta, \Delta' \in \boldsymbol{\Delta}(S)$ are isomorphic, and we can write
$$\boldsymbol{\Delta}(S) \;=\; \{\sigma\Delta \mid \sigma \in G\} ,$$
where $G \subseteq \mathcal{I}so$ is a set of isomorphisms and $\Delta \in \boldsymbol{\Delta}(S)$ is an arbitrarily chosen update set.

(ii) For every state S with $\boldsymbol{\Delta}(S) \;=\; \{\sigma_i\Delta_0 \mid 0 \leq i \leq k\}$ ($G = \{\sigma_0, \ldots, \sigma_k\} \subseteq \mathcal{I}so$) and the corresponding successor states $S_i = S + \sigma_i\Delta_0$ we have
$$\boldsymbol{\Delta}(S_i) \;=\; \sigma_i\boldsymbol{\Delta}(S_0) .$$

Theorem 3. *Locally insignificant choice Abstract State Machines define insignificant choice algorithms as defined by the branching time, abstract state, non-deterministic bounded exploration and background postulates and the insignificant choice restriction.*

We cannot yet provide a proof that locally insignificant choice Abstract State Machines capture insignificant choice algorithms, but it seems as plausible as Theorem 2.

3 The Logic of Non-deterministic ASMs

Let us now tend to associated logics. A logic for deterministic ASMs has been developed by Stärk and Nanchen [36] and proven to be complete. It is also described in [16]. We now look into the extension for non-deterministic ASMs [24] and how it can be adapted to capture the insignificant choice.

3.1 Unrestricted Logic

As the logic of non-deterministic ASMs has to deal with update sets, we let the signature contain a static constant symbol c_f for each dynamic function symbol $f \in \Sigma$, i.e. c_f is not dynamic and has arity 0. We also exploit that the base set contains elements that interpret c_f in every state. By abuse of notation we wrote $(c_f)_S = c_f$. Now let X be a second-order variable of arity 3. For a variable assignment ζ we say that $\zeta(X)$ *represents an update set* Δ iff for each $((f, \bar{a}), b) \in \Delta$ we have $(c_f, \bar{a}, b) \in \zeta(X)$ and vice versa. Here we write $\bar{a}$ for n-tuples, where n is the arity of f.

As for the syntax, with this extension the terms of $\mathcal{L}^{nd}$ are ASM terms. The formulae of the logic are defined inductively as follows:

- If t and t' are terms, then $t = t'$ is a formula.
- If X is an n-ary second-order variable and $t_1, \ldots, t_n$ are terms, then $X(t_1, \ldots, t_n)$ is a formula.
- If r is an ASM rule and X is a second-order variable of arity 3, then $\mathrm{upd}_r(X)$ is a formula.
- If φ and ψ are formulae, then also $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\varphi \rightarrow \psi$ are formulae.
- If φ is a formula, x is a first-order variable and X is a second-order variable, then also $\forall x.\varphi$, $\exists x.\varphi$, $\forall X.\varphi$, $\exists X.\varphi$ are formulae.
- If φ is a formula and X is a second-order variable of arity 3, then $[X]\varphi$ is formula.

The semantics is defined for Henkin structures. A *Henkin prestructure* $\tilde{S}$ over signature Υ is a structure S over Σ with base set B together with sets of relations $D_n \subseteq \mathcal{P}(B^n)$ for all $n \geq 1$.

As the logic uses second-order variables we need extended variable assignments ζ into a Henkin prestructure. For first-order variables x we have $\zeta(x) \in B$ as usual, but for second-order variables X of arity n we request $\zeta(X) \in D_n$. Then with respect to a Henkin prestructure $\tilde{S}$ and such a variable assignment terms are interpreted as usual. The interpretation $[\![\varphi]\!]_{\tilde{S},\zeta}$ for formulae φ is mostly standard with the non-standard parts defined as follows:

- If φ has the form $\forall X.\psi$ with a second-order variable X of order n, then

$$[\![\varphi]\!]_{\tilde{S},\zeta} = \begin{cases} \mathbf{T} & \text{if } [\![\psi]\!]_{\tilde{S},\zeta[X \mapsto A]} = \mathbf{T} \text{ for all } A \in D_n \\ \mathbf{F} & \text{else} \end{cases}.$$

– If φ has the form $[X]\psi$, then

$$[\![\varphi]\!]_{\tilde{S},\zeta} = \begin{cases} \mathbf{F} & \text{if } \mathrm{val}_{S,\zeta}(X) \text{ represents a consistent update set } \Delta \\ & \text{with } [\![\psi]\!]_{\tilde{S}+\Delta,\zeta} = \mathbf{F} \\ \mathbf{T} & \text{else} \end{cases} .$$

While this interpretation is defined for arbitrary Henkin prestructures, it makes sense to restrict the collections D_n of n-ary relations to those that are closed under definability, which defines the notion of Henkin structure. We then say that a sentence is *valid* iff it is interpreted as 1 (i.e., true) in all Henkin structures.

A *Henkin structure* over signature Σ is a Henkin prestructure $\tilde{S} = (S, \{D_n\}_{n\geq 1})$ that is closed under definability, i.e. for every formula φ, every variable assignment ζ and every $n \geq 1$ we have

$$\{(a_1, \ldots, a_n) \in B^n \mid [\![\varphi]\!]_{\tilde{S},\zeta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]} = \mathbf{T}\} \in D_n.$$

3.2 Capturing Insignificant Choice

We now approach a characterisation of the semantic insignificant choice restriction in the logic $\mathcal{L}^{nd}$ defined above. We use isUSet(X) to express that X represents an update set, and conUSet(X) to express that it is consistent—these are defined in [24].

Let us assume that the base set B is defined as the set of hereditarily finite sets $B = HF(A)$ over a finite set A of *atoms*. Then we can express that X is an isomorphism by

$$\begin{aligned} \mathrm{iso}(X) \equiv\ & \forall x, y_1, y_2.(X(x, y_1) \wedge X(x, y_2) \to y_1 = y_2) \wedge \\ & \forall x_1, x_2, y.(X(x_1, y) \wedge X(x_2, y) \to x_1 = x_2) \wedge \forall x \exists y.X(x, y) \wedge \forall y \exists x.X(x, y) \wedge \\ & \bigwedge_{f \in \Upsilon_{dyn}} X(c_f, c_f) \wedge \forall x, y.\Big[X(x, y) \to (x \in \mathit{Atoms} \leftrightarrow y \in \mathit{Atoms}) \wedge \\ & \qquad \forall u.(u \in x \to \exists v.v \in y \wedge X(u, v)) \wedge \forall v.(v \in y \to \exists u.u \in x \wedge X(u, v))\Big] \end{aligned}$$

This leads to the following *insignificance constraint* for a rule r expressing that any two update sets yielded by r are isomorphic:

$$\begin{aligned} \forall X_1, X_2.\ & \mathrm{upd}_r(X_1) \wedge\ \mathrm{upd}_r(X_2) \to \\ & \exists X.(\mathrm{iso}(X) \wedge \mathrm{updIso}(X_1, X_2, X) \wedge \mathrm{updIsoSet}(X_1, X_2, X)) \end{aligned}$$

with

$$\mathrm{updIso}(X_1, X_2, X) \equiv \bigwedge_{f \in \Upsilon_{dyn}} [\forall \bar{x}_1, x_2, \bar{y}_1, y_2.$$
$$(X_1(c_f, \bar{x}_1, x_2) \wedge \bigwedge_{1 \le i \le ar(f)} X(x_{1i}, y_{1i}) \wedge X(x_2, y_2) \rightarrow X_2(c_f, \bar{y}_1, y_2)) \wedge$$
$$\forall \bar{x}_1, x_2, \bar{y}_1, y_2.(X_2(c_f, \bar{x}_1, x_2) \wedge \bigwedge_{1 \le i \le ar(f)} X(x_{1i}, y_{1i}) \wedge X(x_2, y_2) \rightarrow X_1(c_f, \bar{y}_1, y_2))]$$

and

$$\mathrm{updIsoSet}(X_1, X_2, X) \equiv \forall Y_1, Y_2.(\mathrm{isUSet}(Y_1) \wedge \mathrm{isUSet}(Y_2) \wedge \mathrm{updIso}(Y_1, Y_2, X)) \rightarrow ([X_1]\mathrm{upd}_r(Y_1) \leftrightarrow [X_2]\mathrm{upd}_r(Y_2))$$

We can use this characterisation of insignificant choice to modify the logic in such a way that a choice rule will either become an insignificant choice or interpreted as **skip**. For this recall the axiomatic definition of $\mathrm{upd}_r(X)$ from [24]. In order to express insignificant choice we introduce new formulae of the form $\mathrm{upd}_r^{ic}(X)$. If r is not a choice rule, we simply keep the definitions replacing upd by upd^{ic}. For a choice rule r of the form **choose** $v \in \{x \mid x \in Atoms \wedge x \in t\}$ **do** $r'(v)$ **enddo** we define

$$\mathrm{upd}_r^{ic}(X) \leftrightarrow \exists v.v \in Atoms \wedge v \in t \wedge \mathrm{upd}_{r'(v)}^{ic}(X) \wedge$$
$$\forall Y.(\exists x.x \in Atoms \wedge x \in t \wedge \mathrm{upd}_{r'(x)}^{ic}(Y)) \rightarrow$$
$$\exists Z.(\mathrm{iso}(Z) \wedge \mathrm{updIso}(X, Y, Z) \wedge \mathrm{updIsoSet}(X, Y, Z))$$

4 Complexity Restriction

Let us finally look at the link to complexity theory. We define PTIME restricted versions of parallel ASMs [6] and locally insignificant choice [33], which define choiceless polynomial time (CPT) and insignificant choice polynomial time. The former one is strictly included in PTIME; the latter one captures PTIME.

4.1 Choiceless Polynomial Time

In order to define a polynomial time bound on an ASM we have to count steps of a run. If we only take the length of a run, each step would be a macrostep that involves many elementary updates, e.g. the use of unbounded parallelism does not impose any restriction on the number of updates in an update set employed in a transition from one state to a successor state. So we better take the size of update sets into account as well. If objects are sets, their size also matters in estimating what an appropriate microstep is. This leads to the notion of PTIME bound from CPT [6].

A *PTIME (bounded) ASM* is a triple $\tilde{M} = (M, p(n), q(n))$ comprising an ASM M and two integer polynomials $p(n)$ and $q(n)$. A *run* of $\tilde{M}$ is an initial

segment of a run of M of length at most $p(n)$ and a total number of at most $q(n)$ active objects, where n is the size of the input in the initial state of the run.

We say that a PTIME ASM $\tilde{M}$ *accepts* the input structure I iff there is a run of $\tilde{M}$ with initial state generated by I and ending in a state in which *Halt* holds and the value of *Output* is 1. Analogously, a PTIME ASM $\tilde{M}$ *rejects* the input structure I iff there is a run of $\tilde{M}$ with initial state generated by I and ending in a state in which *Halt* holds and the value of *Output* is 0.

A logic $\mathcal{L}$ can be defined by a pair (*Sen*,*Sat*) of functions satisfying the following conditions:

- *Sen* assigns to every signature Σ a recursive set $Sen(\Sigma)$, the set of $\mathcal{L}$*-sentences of signature* Σ.
- *Sat* assigns to every signature Σ a recursive binary relation Sat_Σ over structures S over Σ and sentences $\varphi \in Sen(\Sigma)$. We assume that $Sat_\Sigma(S,\varphi) \Leftrightarrow Sat_\Sigma(S',\varphi)$ holds, whenever S and S' are isomorphic.

We say that a structure S over Σ *satisfies* $\varphi \in Sen(\Sigma)$ (notation: $S \models \varphi$) iff $Sat_\Sigma(S,\varphi)$ holds.

If $\mathcal{L}$ is a logic in this general sense, then for each signature Σ and each sentence $\varphi \in Sen(\Sigma)$ let $K(\Sigma,\varphi)$ be the class of structures S with $S \models \varphi$. We then say that $\mathcal{L}$ is a *PTIME logic*, if every class $K(\Sigma,\varphi)$ is PTIME in the sense that it is closed under isomorphisms and there exists a PTIME Turing machine that accepts exactly the standard encodings of ordered versions of the structures in the class.

We further say that a logic $\mathcal{L}$ *captures PTIME* iff it is a PTIME logic and for every signature Σ every PTIME class of Σ-structures conincides with some class $K(\Sigma,\varphi)$.

4.2 Insignificant Choice Polynomial Time

An *insignificant choice ASM* (for short: icASM) is an ASM M such that for every run $S_0,\ldots,S_k$ of length k such that *Halt* holds in S_k, every $i \in \{0,\ldots,k-1\}$ and every update set $\Delta \in \boldsymbol{\Delta}(S_i)$ there exists a run $S_0,\ldots,S_i,S'_{i+1},\ldots,S'_m$ such that $S'_{i+1} = S_i + \Delta$, *Halt* holds in S'_m, and *Output* = **true** (or **false**, respectively) holds in S_k iff *Output* = **true** (or **false**, respectively) holds in S'_m.

A *PTIME (bounded) insignificant choice ASM* (for short: PTIME icASM) is a triple $\tilde{M} = (M, p(n), q(n))$ comprising an icASM M and two integer polynomials $p(n)$ and $q(n)$ with runs such that whenever an input structure I is accepted by $\tilde{M}$ (or rejected, respectively) then every run on input structure I is accepting (or rejecting, respectively).

According to this definition whenever there exists an accepting or rejecting run, then all other runs on the same input structure, i.e. runs that result making different choices, are also accepting or rejecting, respectively.

Theorem 4. *ICPT captures PTIME on arbitrary finite structures, i.e. ICPT = PTIME.*

The full proof is given in [33]. In a nutshell, given a PTIME problem we simply use a non-deterministic ASM to first generate an order on the set of atoms, then create deterministically the standard encoding of the input structure with this order and finally simulate the PTIME Turing machine deciding the problem. Then it is clear that the choices in the ASM will only refer to atoms, and the local insignificance will be satisfied. This is then used to prove also the converse by creating a PTIME simulation by a Turing machine. The local insignificance condition implies global insignificance, i.e. any choice can be replaced by a fixed choice of the smallest element, and the fact that choices are restricted to atoms guarantees that the local insignificance condition can be checked on a Turing machine in polynomial time. The first part of the proof further shows that PTIME is included in a fragment of ICPT defined by ASMs satisfying the local insignificance condition.

Corollary 1. *PTIME is captured by the fragment $ICPT_{loc}$ of ICPT, where the separating icASM satisfies the local insignificance condition.*

Through Theorem 4 and Corollary 1 ICPT highlights the similarities and differences between classical computation theory on strings using Turing machines and computation theory on structures using ASMs. Not only does the shift to arbitrary Tarski structures lead to a theory on arbitrary level of abstraction, while at the same time enabling the proofs of long-standing open problems such as the refutation of Gurevich's conjecture and the separation of PTIME from NP [33], it shows that computation theory requires more than just functions from input to output. Furthermore, it helps closing the gap between the theory of computation and the developments in practice with the perspective to obtain a thorough theoretical penetration of practice, which is what actually was claimed by the term "Software Engineering".

5 Concluding Remarks

Monographs written or co-authored by Egon Börger provide cornerstones for the development of the theory of computation and its applications [8,11,16,37]. In this article I outlined bits of a modernised theory of computation on structures grounded in behavioural theories of classes of algorithmic systems, associated logics and complexity theory. The emphasis was on polynomial time computations. Starting from parallel algorithms I showed how to extend them by insignificant choice, which requires a modification of the logic of non-deterministic ASMs. Then I sketched the recently proven capture of PTIME. This shows how all parts of the theory fit neatly together. Nonetheless, there are still many open problems associated with the theory, which need to be addressed such as a theory of non-determinism and randomness. The next decisive monograph will be a consolidated theory of computations on structures.

Computations on structures give rise to specification of algorithmic systems on arbitrary levels of abstraction, i.e. they directly feed into rigorous system development. The logics associated with a particular class of algorithmic systems can be used in this context for the verification of desirable properties. Complexity classes enable fine-tuned classification with further insights how an algorithm solving a problem in a complexity class looks like. The various successful applications of the ASM method with or without choice, with single or multiple machines and with sophisticated refinement strategies show that computation theory on structures is well positioned to bridge the gap between the increasing structural complexity of modern software-centric systems and the foundational theory.

References

1. Abiteboul, S., Papadimitriou, C.H., Vianu, V.: The power of reflective relational machines. In: Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS 1994), pp. 230–240. IEEE Computer Society (1994)
2. Abiteboul, S., Vardi, M.Y., Vianu, V.: Fixpoint logics, relational machines, and computational complexity. J. ACM **44**(1), 30–56 (1997). https://doi.org/10.1145/256292.256295
3. Abiteboul, S., Vianu, V.: Generic computation and its complexity. In: Koutsougeras, C., Vitter, J.S. (eds.) Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1991), pp. 209–219. ACM (1991)
4. Arvind, V., Biswas, S.: Expressibility of first order logic with a nondeterministic inductive operator. In: Brandenburg, F.J., Vidal-Naquet, G., Wirsing, M. (eds.) STACS 1987. LNCS, vol. 247, pp. 323–335. Springer, Heidelberg (1987). https://doi.org/10.1007/BFb0039616
5. Blass, A., Gurevich, Y.: Abstract State Machines capture parallel algorithms. ACM Trans. Comput. Logic **4**(4), 578–651 (2003)
6. Blass, A., Gurevich, Y., Shelah, S.: Choiceless polynomial time. Ann. Pure Appl. Logic **100**, 141–187 (1999)
7. Blass, A., Gurevich, Y., Shelah, S.: On polynomial time computation over unordered structures. J. Symbol. Logic **67**(3), 1093–1125 (2002)
8. Börger, E.: Computability, Complexity, Logic, Studies in Logic and the Foundations of Mathematics, vol. 128. North-Holland (1989)
9. Börger, E.: The ASM refinement method. Formal Aspects Comput. **15**(2–3), 237–257 (2003). https://doi.org/10.1007/s00165-003-0012-7
10. Börger, E., Durdanovic, I.: Correctness of compiling Occam to Transputer code. Comput. J. **39**(1), 52–92 (1996). https://doi.org/10.1093/comjnl/39.1.52
11. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic. Springer, Heidelberg (1997)
12. Börger, E., Raschke, A.: Modeling Companion for Software Practitioners. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56641-1
13. Börger, E., Rosenzweig, D.: A mathematical definition of full Prolog. Sci. Comput. Program. **24**(3), 249–286 (1995). https://doi.org/10.1016/0167-6423(95)00006-E
14. Börger, E., Schewe, K.D.: Concurrent abstract state machines. Acta Informatica **53**(5), 469–492 (2016). https://doi.org/10.1007/s00236-015-0249-7
15. Börger, E., Schewe, K.D.: A behavioural theory of recursive algorithms. Fundamenta Informaticae **177**(1), 1–37 (2020)

16. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18216-7
17. Chandra, A.K., Harel, D.: Structure and complexity of relational queries. J. Comput. Syst. Sci. **25**(1), 99–128 (1982). https://doi.org/10.1016/0022-0000(82)90012-5
18. Dawar, A., Richerby, D.: Fixed-point logics with nondeterministic choice. J. Log. Comput. **13**(4), 503–530 (2003). https://doi.org/10.1093/logcom/13.4.503
19. Dawar, A., Richerby, D., Rossman, B.: Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs. Ann. Pure Appl. Log. **152**(1–3), 31–50 (2008). https://doi.org/10.1016/j.apal.2007.11.011
20. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Perspectives in Mathematical Logic. Springer, Heidelberg (1995). https://doi.org/10.1007/978-3-662-03182-7
21. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R. (ed.) SIAM-AMS Proceedings, pp. 43–73, no. 7 (1974)
22. Ferrarotti, F., Schewe, K.D., Tec, L., Wang, Q.: A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. Theoret. Comput. Sci. **649**, 25–53 (2016). https://doi.org/10.1016/j.tcs.2016.08.013
23. Ferrarotti, F., Schewe, K.D., Tec, L., Wang, Q.: A complete logic for Database Abstract State Machines. Logic J. IGPL **25**(5), 700–740 (2017)
24. Ferrarotti, F., Schewe, K.D., Tec, L., Wang, Q.: A unifying logic for non-deterministic, parallel and concurrent Abstract State Machines. Ann. Math. Artif. Intell. **83**(3–4), 321–349 (2018). https://doi.org/10.1007/s10472-017-9569-3
25. Gire, F., Hoang, H.K.: An extension of fixpoint logic with a symmetry-based choice construct. Inf. Comput. **144**(1), 40–65 (1998). https://doi.org/10.1006/inco.1998.2712
26. Gurevich, Y.: A new thesis (abstract). Am. Math. Soc. **6**(4), 317 (1985)
27. Gurevich, Y.: Logic and the challenge of computer science. In: Börger, E. (ed.) Current Trends in Theoretical Computer Science, pp. 1–57. Computer Science Press (1988)
28. Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In: Börger, E. (ed.) Specification and Validation Methods, pp. 9–36. Oxford University Press (1995)
29. Gurevich, Y.: Sequential abstract state machines capture sequential algorithms. ACM Trans. Comput. Logic **1**(1), 77–111 (2000)
30. Immerman, N.: Descriptive Complexity. Graduate texts in Computer Science. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0539-5
31. Libkin, L.: Elements of Finite Model Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07003-1
32. Naur, P., Randell, B.: Software Engineering (1968). Report on a conference sponsored by the NATO Science Committee
33. Schewe, K.D.: Insignificant choice polynomial time. CoRR abs/2005.04598 (2021). http://arxiv.org/abs/2005.04598
34. Schewe, K.D., Ferrarotti, F.: Behavioural theory of reflective algorithms I: reflective sequential algorithms. CoRR abs/2001.01873 (2020). http://arxiv.org/abs/2001.01873
35. Schewe, K.-D., Ferrarotti, F.: A logic for reflective ASMs. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 93–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_7
36. Stärk, R., Nanchen, S.: A logic for abstract state machines. J. Univ. Comput. Sci. **7**(11) (2001)

37. Stärk, R.F., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-642-59495-3
38. Van den Bussche, J., Van Gucht, D.: Semi-determinism. In: Vardi, M.Y., Kanellakis, P.C. (eds.) Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 191–201. ACM Press (1992). https://doi.org/10.1145/137097.137866