# Distributed Adaptive Systems

## Theory, Specification, Reasoning

**Klaus-Dieter Schewe, Flavio Ferrarotti, Loredana Tec, Qing Wang**

Linz, Austria & Canberra, Australia

kdschewe@acm.org

# 1 Questions that Need Answers

- **Expressiveness** of Rigorous Methods
  - Can we give a precise characterisation of a class $\mathcal{C}$ of systems that are captured by our method $\mathcal{M}$?
  - If we capture $\mathcal{C}' \subsetneq \mathcal{C}$, can we precisely characterise what we gain (e.g. easier proofs, reduced complexity, etc.) for the reduced expressiveness?
  - Can we justify that $\mathcal{C}' \subsetneq \mathcal{C}$ is of equal importance as $\mathcal{C}$ itself?
  - Can we ensure that our method adequately captures the technology and implementation languages that are commonly used for implementations of the captured class $\mathcal{C}$ of systems?
  - Can we ensure that our refinement-based development methodology provides an adequate (in terms of effort, feasibility, quality of the result) for the development of systems in the class of interest?

# Questions / cont.

- **Logical Reasoning** with Rigorous Methods
  - Can we provide a logic for our method by means of which all desirable properties of a system of interest can be expressed?
  - Can we provide a (preferably complete) proof theory for such a logic that enables mechanical reasoning complementing proofs by brain and pencil?
  - Can we ensure that proofs can exploit previous knowledge?
  - Can we provide pragmatic guidance for conducting proofs?
  - Can we provide pragmatic refinement rules that have been proven a priori to be correct?

# Behavioural Systems Theory

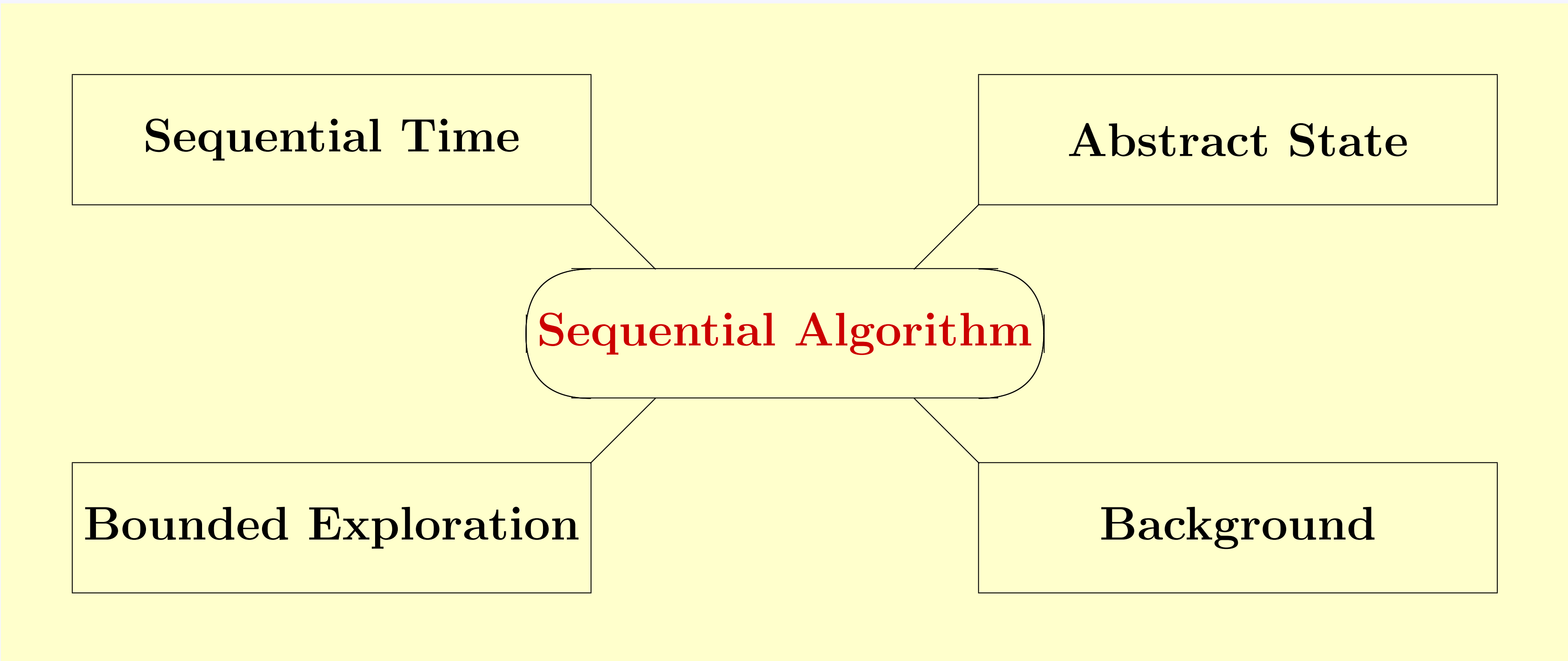
## Foundations for Expressiveness and Logical Reasoning:

- Provide a language-independent definition of a *class $\mathcal{C}$ of systems of interest*
- Provide an *abstract machine model $\mathcal{M}$* (i.e. the rigorous method)
- Prove that the abstract machine model $\mathcal{M}$ *captures* the class $\mathcal{C}$ of systems
  - **Plausibility:** Show the satisfaction of the characterising properties of $\mathcal{C}$
  - **Capture:** Show that every system stipulated by the characterising properties of $\mathcal{C}$ can be specified by a *behaviourally equivalent* abstract machine
- Derive a *logic* $\mathcal{L}$ from $\mathcal{M}$ and show how to express desirable properties of systems in $\mathcal{C}$ in this logic

# 2 Foundations: Expressiveness

Ur-Instance: **Behavioural Theory of Sequential Algorithms** (aka: Sequential ASM Thesis)

***Sequential algorithm***s are defined by four conditions (***postulates***):



# Characterising Postulates

- **Sequential Time:** A sequential algorithm $t$ is associated with a non-empty set of states $\mathcal{S}_t$, a subset $\mathcal{I}_t \subseteq \mathcal{S}_t$ of initial states, and a transition function $\tau_t : \mathcal{S}_t \to \mathcal{S}_t$.
  - This already determines the notion of a ***run*** $S_0, S_1, \dots$ with $S_0 \in \mathcal{I}_t$ and $S_{i+1} = \tau_t(S_i)$
- **Abstract State:** All states $S \in \mathcal{S}_t$ of a sequential algorithm $t$ are structures over the same signature $\Sigma_t$. For all states $S$ and $\tau_t(S)$ have the same base set $B$. The sets of states and initial states are closed under isomorphisms.
- **Bounded Exploration:** For a sequential algorithm $t$ there exists a fixed, finite set $W$ (***bounded exploration witness***) of ground terms such that $\Delta(t, S_1) = \Delta(t, S_2)$ holds whenever the states $S_1$ and $S_2$ coincide over $W$.
- **Background:** There exists a background class providing at least truth values and their junctors and a value *undef*.

Here $\Delta(t, S)$ is the uniquely defined ***update set*** of the algorithm $t$ in state $S$, i.e. the set of updates $(\ell, v)$ defined by the transition from $S$ to $\tau_t(S)$.

# Sequential ASM Rules

- ***Sequential ASM rules*** over a signature $\Sigma$ are defined as follows:
  - If $t_0, \dots, t_n$ are terms over $\Sigma$, and $f$ is a $n$-ary function symbol in $\Sigma$, then $f(t_1, \dots, t_n) := t_0$ is a rule $r$ in $\mathcal{R}$ called ***assignment rule***.
  - If $\varphi$ is a Boolean term and $r' \in \mathcal{R}$ is a DB-ASM rule, then **if** $\varphi$ **then** $r'$ **endif** is a rule $r$ in $\mathcal{R}$ called ***conditional rule***.
  - If $r_1, \dots, r_n$ are rules in $\mathcal{R}$, then the rule $r$ defined as **par** $r_1 \dots r_n$ **endpar** is a rule in $\mathcal{R}$, called ***parallel rule***.
- Each rule $r$ yields an update set $\Delta(r, S)$ for a state $S$ over $\Sigma$.

# Sequential Abstract State Machines

**Definition.** A ***sequential Abstract State Machine*** (ASM) $\mathcal{M}$ over a signature $\Sigma$ consists of

- a set $\mathcal{S}_{\mathcal{M}}$ of states over $\Sigma$ and a non-empty subset $\mathcal{I}_{\mathcal{M}} \subseteq \mathcal{S}_{\mathcal{M}}$ of initial states, both closed under isomorphisms,
- a closed sequential **ASM rule** $r_{\mathcal{M}}$ over $\Sigma$, and
- a function $\tau_{\mathcal{M}} : \mathcal{S}_{\mathcal{M}} \to \mathcal{S}_{\mathcal{M}}$ determined by $r_{\mathcal{M}}$ such that $\tau_{\mathcal{M}}(S) = S + \Delta(r_{\mathcal{M}}, S)$ holds.

**Theorem (Plausibility Theorem).** Each sequential ASM $\mathcal{M}$ defines a sequential algorithm with the same signature as $\mathcal{M}$.

**Theorem (Characterisation Theorem).** For every sequential algorithm there exists a behaviourally equivalent sequential ASM $\mathcal{M}$.

Y. Gurevich, Sequential abstract-state machines capture sequential algorithms, ACM Trans. Comput. Log. 1 (1) (2000): 77-111.

# Proof Sketch

1. Fix a bounded exploration witness $W$ (w.l.o.g. closed under subterms)
2. Take a state $S$ and an update $((f,(v_1,\dots,v_n)),v_0) \in \Delta(S)$
3. Show that each $v_i$ is a *critical value* in $S$, i.e. it results from interpretation of a ground term $t_i \in W$ in the state $S$
4. Then the update is yielded by the rule $f(t_1,\dots,t_n) := t_0$, and consequently $\Delta(S)$ results from a rule $r_S$ that is the parallel composition of such assignments
5. Generalise to states that coincide on $W$: If $S'$ and $S$ coincide on $W$, then $\Delta(S') = \Delta(r_S, S')$
6. Extend to isomorphic states: If $S_1, S_2$ are isomorphic and $\Delta(S_1) = \Delta(r_S, S_1)$, then also $\Delta(S_2) = \Delta(r_S, S_2)$ holds
7. Define $W$-similarity: States $S_1, S_2$ are *$W$-similar* iff $val_{S_1}(t_i) = val_{S_1}(t_j) \Leftrightarrow val_{S_2}(t_i) = val_{S_2}(t_j)$ holds for all $t_i, t_j \in W$
8. Extend to $W$-similar states: If $S'$ and $S$ are $W$-similar, then $\Delta(S') = \Delta(r_S, S')$
9. As there are only finitely many $W$-equivalence classes, use conditional rules to finally create a rule $r$ with $\Delta(S') = \Delta(r, S')$ for all states $S'$

# Easy Observations

- The proof mainly exploits properties derived from the postulates

- Any other method providing parallel assignments and guards could have been used in the proof as well

- An extension to cover *bounded non-determinism* is also straightforward:

  - In the sequential time postulate replace the state transition function $\tau_t$ by a relation

  - Add a bounded choice rule to sequential ASMs

## 2.1 Extension: Unbounded (Synchronous) Parallelism

- The parallel "branches" involved in a single step do not only depend on the algorithm, but also on the state
  - The key is to exploit **multiset comprehension terms** (instead of just ground terms) in the bounded exploration postulate (conjecture launched at ABZ 2012)
  - In addition, the background structure in the background postulate must provide constructors for tuples and multisets together with the corresponding operators on them
- Sequential ASMs need to be extended to *parallel ASMs* that exploit a general **forall**-rule for unbounded parallelism:
  - If $\varphi$ is a term with $\{x_1, \dots, x_k\} \subseteq fr(\varphi)$ and $r' \in \mathcal{R}$ is an ASM rule, then **forall** $x_1, \dots, x_k$ **with** $\varphi$ **do** $r'$ **enddo** is a rule $r$ in $\mathcal{R}$

# Proof Sketch

Only Step 4 in the previous proof (trivial for sequential algorithms) requires an amendment, the rest remains more or less the same

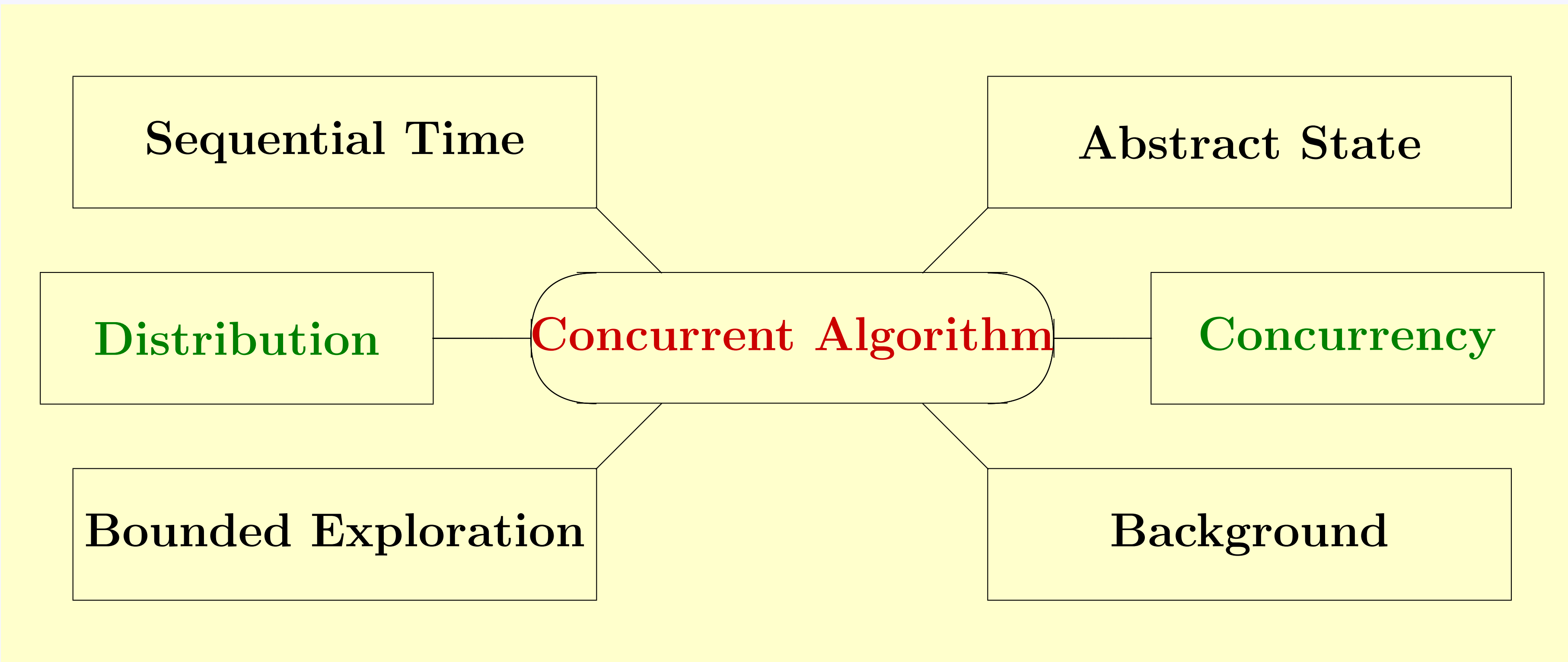
1. Define a logical theory (derived from $W$) and show that whenever tuple $\bar{a} = (a_0, \dots, a_r)$ and $\bar{b} = (b_0, \dots, b_r)$ have the same type (i.e. they satisfy exactly the same formulae in this theory), then $((f, (a_1, \dots, a_r), a_0)$ appears in an update set $\Delta(S)$ iff $((f, (b_1, \dots, b_r), b_0) \in \Delta(S)$.
2. For this assume first that there is an isomorphism taking $\bar{a}$ to $\bar{b}$ and apply the bounded exploration postulate, then use a Gödelisation to tackle the general case.
3. Show that for the theory isolating formulae exist, i.e. tuples have the same type iff they are satisfied by the isolating formula of the type.
4. Use the isolating formula to define a rule

$$\textbf{forall } x_0, x_1, \dots, x_r \textbf{ with } t_\chi^{\bar{a}}(x_0, x_1, \dots, x_r) \textbf{ do } f(x_1, \dots, x_r) := x_0$$

F. Ferrarotti, K.-D. Schewe, L. Tec, Q. Wang: A New Thesis concerning Synchronised Parallel Computing – Simplified Parallel ASM Thesis. Theor. Comp. Sci. 649 (2016): 25-53.

## 2.2 Extension: (Asynchronous) Concurrency

***Concurrent algorithms*** require two additional ***postulates***:



# Characterising Postulates

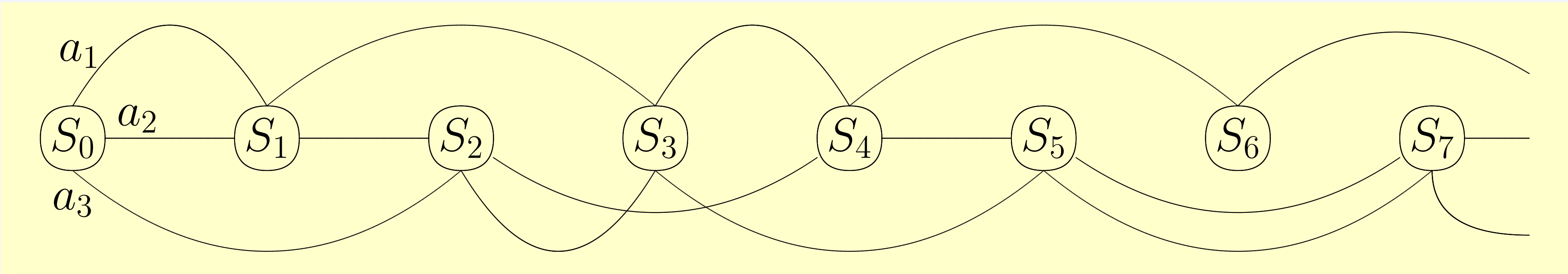
- **Distribution:** A *distributed (adaptive) system* (DAS) is given by a set $\mathcal{A}$ of agents $a$, each equipped with a parallel (reflective) algorithm $alg(a)$. Furthermore, there is a set $\mathbb{L}$ of localities and and assignment $loc : \mathcal{A} \rightarrow \mathbb{L}$.

- **Concurrency:** A DAS $\mathcal{D} = \{(a, alg(a)) \mid a \in \mathcal{A}\}$ defines *concurrent $\mathcal{D}$-runs* $S_0, S_1, \dots$ starting in some initial state $S_0$, such that each state $S_n$ ($n \geq 0$) yields a next state $S_{n+1}$ by a finite set $\mathcal{A}_n$ of agents simultaneously completing the execution of their current $alg(a)$-step they had started in some preceding state $S_j$ ($j \leq n$ depending on $a$), i.e. $S_{n+1} = S_n + \bigcup_{a \in \mathcal{A}_n} \Delta_a(S_j)$.

  - Informally phrased, in a concurrent run the sequence of states results from simultaneously applying update sets of several individual machines that have been built on previous (not necessarily the last nor the same) states.

  - Concurrent runs do not rely on interleaving, but permit simultaneous updates by several machines

# Concurrent ASMs Capture Concurrent Algorithms




- Simply use families of ASMs indexed by agents, i.e. $\{(a, \mathcal{M}_a) \mid a \in \mathcal{A}\}$
- Exploit that locations between different ASMs in the family can be shared
- Define concurrent runs in analogy to the concurrency postulate
- The proof that concurrent ASMs capture concurrent algorithms is straightforward: reduction to the already known proofs

E. Börger, K.-D. Schewe: Concurrent Abstract State Machines. Acta Inform. 53 (5), (2016): 469-492 (open access).

## 2.3 Extension: Adaptivity through Linguistic Reflection

- ***Reflection:*** each agent computes not only updates to the state, but also to itself, which requires (for each agent) a function from pairs of states and specifications to specifications:
  - Think of pairs $(S_i, P_j)$ comprising a state $S_i$ (as in the sequential thesis), and a (sequential or parallel) algorithm $P_j$
  - Consider transition functions $\tau_j : (S_i, P_j) \mapsto (S_{i+1}, P_j)$ not changing the algorithm $P_j$, and transition functions $\sigma_i : (S_i, P_j) \mapsto (S_i, P_{j+1})$ changing only the algorithm
  - A run of a reflective algorithm corresponds to the sequence of pairs $(S_i, P_i)$, where in each step both the state $S_i$ and the algorithm $P_i$ are updated
- This requires changes to all postulates; the key issue is to permit *terms as values*

F. Ferrarotti, K.-D. Schewe, L. Tec: A Behavioural Theory for Reflective Sequential Algorithms. Perspectives in Systems Informatics. LNCS vol. 10742, pp. 117-131, Springer 2018.

# Sequential Time

- We can capture the state-algorithm pairs by an extension $\Sigma_{ext}$ of the signature $\Sigma$ using additional function symbols to represent the algorithm, e.g. capturing the signature and some syntactic description

- We must further permit new function symbols to be created, which can be done by exploiting the concept of "reserve"

**Reflective Sequential Time Postulate.**
A ***reflective algorithm*** $\mathcal{A}$ consists of the following:

- A non-empty set $\mathcal{S}_{\mathcal{A}}$ of *extended states*.
- A non-empty subset $\mathcal{I}_{\mathcal{A}} \subseteq \mathcal{S}_{\mathcal{A}}$ of *initial extended states* such that for all $(S, P), (S', P') \in \mathcal{I}_{\mathcal{A}}$, it holds that $S$ and $S'$ are first-order structures of a same signature $\Sigma$ and $P|_{\Sigma}$ and $P'|_{\Sigma}$ have exactly the same runs.
- A *one-step transformation* function $\tau_{\mathcal{A}} : \mathcal{S}_{\mathcal{A}} \rightarrow \mathcal{S}_{\mathcal{A}}$ such that $\tau_{\mathcal{A}}((S, P)) = (S', P')$ iff $\tau_P((S, P)) = (S', P')$ for the one-step transformation function $\tau_P$ of the algorithm $P$.

# Behavioural Equivalence

- While behavioural equivalent sequential/parallel algorithms have exactly the same runs, this is not necessarily the case for reflection

- For runs $r_1 = (S_0, P_0), (S_1, P_1), (S_2, P_2), \dots,$ and $r_2 = (S'_0, P'_0), (S'_1, P'_1), (S'_2, P'_2), \dots,$ defne that $r_1$ and $r_2$ are ***essentially equivalent*** if for every $i \geq 0$ the following holds:

  - $S_i = S'_i$
  - $S_i$ and $S'_i$ are first-order structures of a same signature $\Sigma_i$ and $P_i|_{\Sigma_i}$ and $P'_i|_{\Sigma_i}$ have exactly the same runs

**Definition (Behavioural Equivalence).**
Two reflective algorithms $\mathcal{A}$ and $\mathcal{A}'$ are ***behaviourally equivalent*** iff $\mathcal{A}$ and $\mathcal{A}'$ have essentially equivalent classes of essentially equivalent runs, i.e. there is a bijection $\zeta$ between runs of $\mathcal{A}$ and $\mathcal{A}'$, respectively, such that $r$ and $\zeta(r)$ are essentially equivalent for all run $r$.

# Abstract States

- States are first-order structures, but must also include (an encoding of) an algorithm given by a finite text

- The representation of algorithms in a state requires terms that are used by the algorithms to appear as values. So we have to allow terms over $\Sigma$ (including the dormant function symbols in the reserve) to be at the same time values in an extended base set

- States $(S, P)$ and $(S', P')$ are ***essentially isomorphic*** if $S$ and $S'$ are isomorphic first-order structures of some vocabulary $\Sigma$ and $P|_\Sigma$ and $P'|_\Sigma$ have exactly the same runs

- If $\zeta$ is an isomorphism form $S$ to $S'$, then we say that $(S, P)$ and $(S', P')$ are ***essentially isomorphic via*** $\zeta$

# Extended Abstract States

**Reflective Abstract State Postulate.**
Let $\mathcal{A}$ be a reflective algorithm. Fix a signature $\Sigma$ and an extension $\Sigma_{ext}$ of the signature $\Sigma$ with additional function names.

- States of $\mathcal{A}$ are first-order structures of signature $\Sigma_{ext}$.
- Every state $(S, P)$ of $\mathcal{A}$ is formed by the *disjoint* union of an arbitrary first-order structure $S$ of some *finite* signature $\Sigma_{st} \subseteq \Sigma$ and a first-order structure $S_P$ of signature $\Sigma_{wt} = \Sigma_{ext} \setminus \Sigma$ which contains an encoding of the sequential algorithm $P$.
- The one-step transformation $\tau_{\mathcal{A}}$ of a RSA $\mathcal{A}$ does not change the base set of any state of $\mathcal{A}$.
- The sets $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{I}_{\mathcal{A}}$ of, respectively, states and initial states of $\mathcal{A}$, are closed under essentially isomorphic states.
- If two states $(S, P)$ and $(S', P')$ of $\mathcal{A}$ are essentially isomorphic via an isomorphism $\zeta$ from $S$ to $S'$, then $\tau_{\mathcal{A}}((\mathbf{S}_1, A_1))$ and $\tau_{\mathcal{A}}((\mathbf{S}_2, A_2))$ are also essentially isomorphic via $\zeta$.

# Bounded Exploration

- Each algorithm $P_i$ represented in state $(S_i, P_i)$ has its own bounded exploration witness $W_i$
- For parallel algorithms $P_i$ such a bounded exploration witness is a set of multiset comprehension terms, where each element in such a multiset corresponds to a branch (or proclet) of the parallel computation
- Due to the construction of $W_i$ in the characterisation proof we know that $W_i$ is somehow contained in the finite representation of $P_i$
- E.g., the ASM rule constructed in the proof of the parallel ASM thesis only contains terms derived from the terms in $W_i$, and this holds analogously for any other representation of $P_i$
- Thus, the terms in $W_i$ result by interpretation from terms that appear in the representation of any algorithm, and there must exist a finite set of terms $W$ such that its interpretation in an extended state yields both values and terms, and the latter represent $W_i$
- Consequently, the interpretation of $W$ and of its interpretation in an extended state suffice to determine the update set in that state

# Strong Coincidence

We first need an extension of the notion of **strong coincidence** over a set of multiset comprehension terms

**Definition (Strong Coincidence).**
Let $(S, P)$ and $(S', P')$ be states of signature $\Sigma_{ext}$. Let $W_{st}$ be a set of multiset comprehension terms over signature $\Sigma$ and $W_{wt}$ be a set of multiset comprehension terms over signature $\Sigma_{ext} \setminus \Sigma$. $(S, P)$ and $(S', P')$ ***strongly coincide*** over $W_{st} \cup W_{wt}$ iff the following holds:

- For every $t \in W_{st}$, $val_{(S,P)}(t) = val_{(S',P')}(t)$.
- For every $t \in W_{wt}$,
  - $val_{(S,P)}(t) = val_{(S',P')}(t)$.
  - $val_{(S,P)}(raise(t)) = val_{(S',P')}(raise(t))$,
    where $raise(t)$ denotes the interpretation of $t$ as a term of signature $\Sigma$.

# Reflective Bounded Exploration

- Use $\Delta(P, S)$ to denote the unique set of updates produced by the sequential algorithm $P$ in state $S$
- The unique set of updates produced by a RSA $\mathcal{A}$ in a state $(S, P)$ is defined as $\Delta(\mathcal{A}, (S, P)) = \Delta(P, (S, P))$
- The idea of the modified bounded exploration postulate is that, for every state $(S_i, P_i)$, the multiset comprehension terms obtained by the interpretation in $(S_i, P_i)$ of the terms in $W_{wt}$ together with the "standard" terms in $W_{st}$ form a bounded exploration witness for the sequential algorithm $P_i$

# Reflective Bounded Exploration Postulate

**Reflective Bounded Exploration Postulate.**
For every reflective $\mathcal{A}$ of signature $\Sigma_{ext}$ there is a finite set $W_{st}$ of multiset comprehension terms over signature $\Sigma$ and a finite set $W_{wt}$ of multiset comprehension terms over signature $\Sigma_{ext} \setminus \Sigma$ such that $\Delta(\mathcal{A}, (S, P)) = \Delta(\mathcal{A}, (S', P'))$ holds, whenever states $(S, P)$ and $(S', P')$ of $\mathcal{A}$ strongly coincide on $W_{st} \cup W_{wt}$.

If a set of multiset comprehension terms $W_{st} \cup W_{wt}$ satisfies the reflective bounded exploration postulate, we call it a ***reflective bounded exploration witness* (R-witness)** for $\mathcal{A}$

A ***reflective algorithm*** (RA) is characterised by the Reflective Sequential Time, Reflective Abstract State, Reflective Background and Reflective Bounded Exploration Postulates.

# Background Postulate

- Parallelism and reflection require the presence of additional constructors

**Reflective Background Postulate.**
Let $A$ be a reflective algorithm of *vocabulary* $\Sigma_{ext}$ with background class $\mathcal{K}$. The vocabulary $\Sigma_{\mathcal{K}}$ of $\mathcal{K}$ includes (at least) a binary *tuple constructor* and a *multiset constructor* of unbounded arity; and the vocabulary $\Sigma_{\mathbf{B}}$ of the background of the computation states of $A$ includes (at least) the following *obligatory* function symbols:

- Nullary function symbols `true`, `false`, `undef` and $\oslash$.
- Unary function symbols `reserve`, `atomic`, `Boole`, $\neg$, `first`, `second`, $\{\!\{\cdot\}\!\}$, $\biguplus$ and `AsSet`.
- Binary function symbols $=$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\uplus$ and $(\,,\,)$.
- *raise* mapping terms over $\Sigma_{ext}$ to terms over $\Sigma$.

All function symbols in $\Sigma_{\mathbf{B}}$, with the sole exception of `reserve`, are static.

# Reflective ASMs

- In a **reflective ASM** the following extensions to signature, background and rules apply:
  - The signature contain a function symbol **self** capturing the signature and rule of the actual ASM
  - The background structure captures all constructs required by the reflective background postulate
  - If **self** is to be bound to a tree structure, then the tree operators are defined in the background
- In a run of an individual ASM $asm_a$ in each step always the actual rule in **self** is applied

# Tree Algebra

- An *unranked tree* is a structure $(\mathcal{O}, \prec_c, \prec_s)$ consisting of a finite, non-empty set $\mathcal{O}$ of node identifiers, called *tree domain*, ordering relations $\prec_c$ and $\prec_s$ over $\mathcal{O}$ called *child relation* and *sibling relation*, respectively, satisfying the following conditions:

  - there exists a unique, distinguished node $o_r \in \mathcal{O}$ (called the *root* of the tree) such that for all $o \in \mathcal{O} - \{o_r\}$ there is exactly one $o' \in \mathcal{O}$ with $o' \prec_c o$,

  - whenever $o_1 \prec_s o_2$ holds, then there is some $o \in \mathcal{O}$ with $o \prec_c o_i$ for $i = 1, 2$, and

  - the relations $\prec_c$ and $\prec_s$ are irreflexive $(x \not\prec x)$.

K.-D. Schewe, Q. Wang: XML Database Transformations. J. UCS vol. 16 (20): 3043-3072, 2010

# Trees, Hedges and Contexts

- A *tree* $t$ (over the set of labels $\mathcal{L}$ with values in the universe $U$) is a triple $(\gamma_t, \omega_t, \upsilon_t)$ consisting of an unranked tree $\gamma_t = (\mathcal{O}_t, \prec_c, \prec_s)$, a total *label function* $\omega_t \colon \mathcal{O}_t \to \Sigma$, and a partial *value function* $\upsilon_t \colon \mathcal{O}_t \to U$ such that whenever $\upsilon_t$ is defined on the argument $o$, $o$ is a leaf in $\gamma_t$.

- A sequence $t_1, \dots, t_k$ of trees is called a *hedge*, and a multiset $\{\!\{t_1, \dots, t_k\}\!\}$ of trees is called a *forest* – $\varepsilon$ denotes the *empty hedge*.

- The set of *contexts* over an alphabet $\mathcal{L}$ ($\xi \notin \mathcal{L}$) is the set $T_{\mathcal{L} \cup \{\xi\}}$ of unranked trees over $\mathcal{L} \cup \{\xi\}$ such that for each tree $t \in T_{\mathcal{L} \cup \{\xi\}}$ exactly one leaf node is labelled with the symbol $\xi$ and has undefined value, and all other nodes in a tree are labelled and valued in the same way as a tree.

# Operations on Trees

- ***Tree-to-tree substitution.*** For a tree $t_1 \in T_{\mathcal{L}_1}$ with a node $o \in \mathcal{O}_{t_1}$ and a tree $t_2 \in T_{\mathcal{L}_2}$ the result $t_1[\widehat{o} \mapsto t_2]$ of substituting $t_2$ for the subtree rooted at $o$ is a tree in $T_{\mathcal{L}_1 \cup \mathcal{L}_2}$.

- ***Tree-to-context substitution.*** For a tree $t_1 \in T_{\mathcal{L}_1}$ with a node $o \in \mathcal{O}_{t_1}$ the result $t_1[\widehat{o} \mapsto \xi]$ of substituting the trivial context for the subtree rooted at $o$ is a context in $T_{\mathcal{L}_1 \cup \{\xi\}}$.

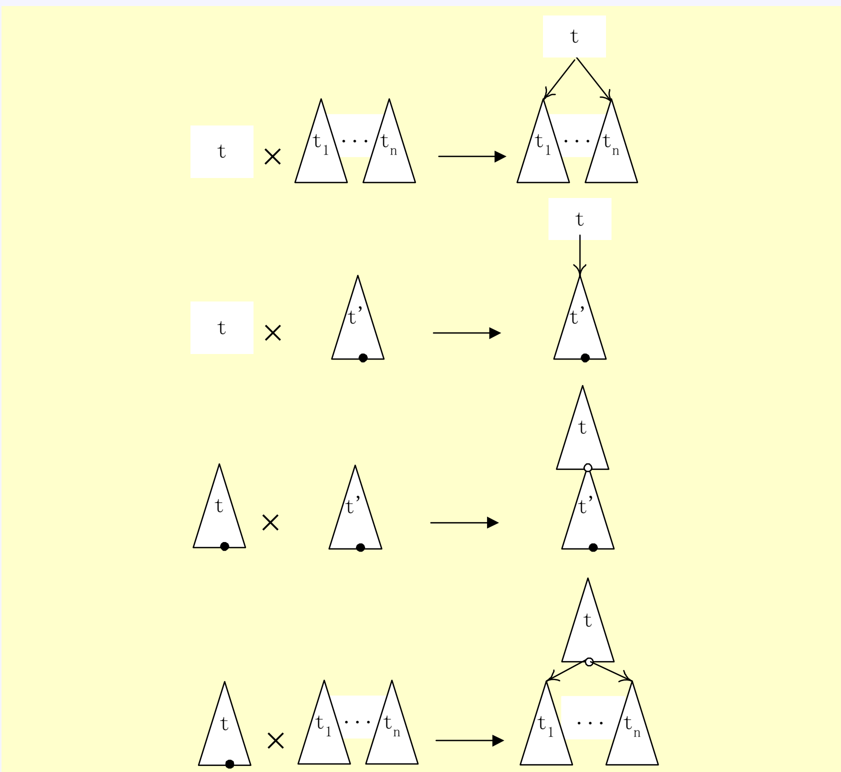
- ***Context-to-context substitution.*** For a context $c_1 \in T_{\mathcal{L}_1 \cup \{\xi\}}$ and a context $c_2 \in T_{\mathcal{L}_2 \cup \{\xi\}}$ the result $c_1[\xi \mapsto c_2]$ of substituting $c_2$ for the node labelled by $\xi$ in $c_1$ is a context in $T_{\mathcal{L}_1 \cup \mathcal{L}_2 \cup \{\xi\}}$.

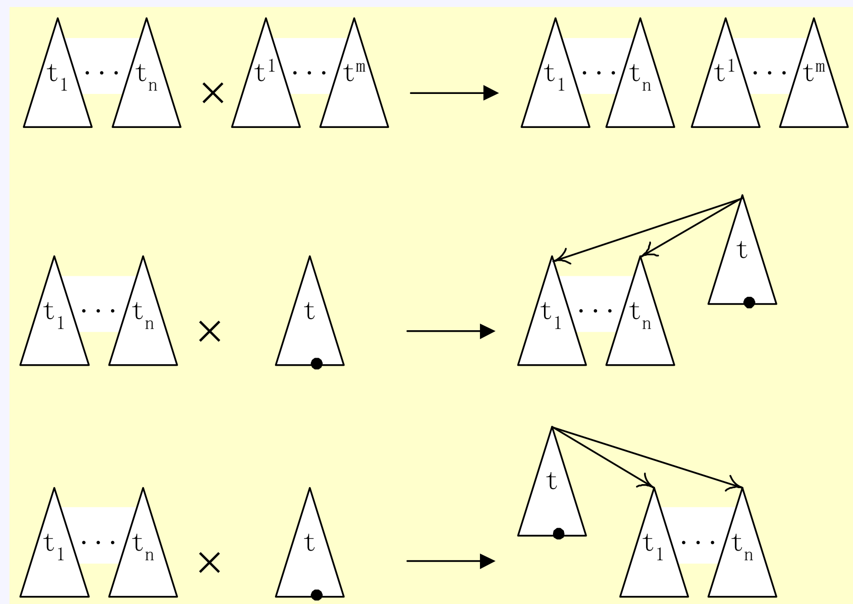
- ***Context-to-tree substitution.*** For a context $c_1 \in T_{\mathcal{L}_1 \cup \{\xi\}}$ and a tree $t_2 \in T_{\mathcal{L}_2}$ the result $c_1[\xi \mapsto t_2]$ of substituting $t_2$ for the node labelled by $\xi$ in $c_1$ is a tree in $T_{\mathcal{L}_1 \cup \mathcal{L}_2}$.

# Further Operations on Trees

- *context* is a binary, partial function defined on pairs $(o_1, o_2)$ of nodes with $o_i \in \mathcal{O}_t$ $(i = 1, 2)$ such that $o_1$ is an ancestor of $o_2$, i.e. $o_1 \prec_c^* o_2$ holds for the transitive closure $\prec_c^*$ of $\prec_c$. We have $context(o_1, o_2) = \widehat{o}_1[\widehat{o}_2 \mapsto \xi]$.

- *subtree* is a unary function defined on $\mathcal{O}_t$. We have $subtree(o) = \widehat{o}$.



# A Glimpse of the Proof

- Define the set of *terms generated by $W_{wt}$ in $(S,P)$* as
$$G_{W_{wt}}^{(S,P)} = \{raise(t') \mid val_{(S,P)}(t) = t' \text{ for some } t \in W_{wt}\}$$
- Show again that every value occurring in an update is ***critical***, i.e. results from the interpretation of the bounded exploration witness terms
- Show again that any tuple with the same logical type as the tuple defined by an update in $\Delta(\mathcal{A},(S,P))$ also gives rise to an update, from which we can conclude again the existence of an ASM rule that yields the update set at hand
- We obtain for every extended state $(S,P)$ a rule $r_{(S,P)}$ such that $r_{(S,P)}$ uses only critical terms and $\Delta(r_{(S,P)},(S,P)) = \Delta(\mathcal{A},(S,P))$ holds
- If two states $(S,P)$ and $(S',P')$ of $\mathcal{A}$ are relative $W[(S,P)]$-equivalent and coincide over $W[(S,P)]$, then it follows that $\Delta_{st}(r_{(S,P)},(S',P')) = \Delta_{st}(\mathcal{A},(S',P'))$
  - Two states $(S_1,P_1)$ and $(S_2,P_2)$ of $\mathcal{A}$ are ***$W$-equivalent relative to $\mathcal{C}[(S,P)]$*** iff $(S_1,P_1),(S_2,P_2) \in \mathcal{C}[(S,P)]$ and $E_{(S_1,P_1)} = E_{(S_2,P_2)}$, where (for $i=1,2$) $E_{(S_i,P_i)}(t_1,t_2) \equiv val_{(S_i,P_i)}(t_1) = val_{(S_i,P_i)}(t_2)$ is an equivalence relation on the set of critical terms of $(S,P)$

# A Glimpse of the Proof (cont.)

- For every class $\mathcal{C}([S_i, P_i])$ of states of $\mathcal{A}$, we have a corresponding rule $r_{[(S_i,P_i)]}$ with
  - $\Delta_{st}(r_{[(S,P)]}, (S_i, P_i)) = \Delta_{st}(\mathcal{A}, (S_i, P_i))$ for every state $(S_i, P_i) \in \mathcal{C}[(S, P)]$, i.e., for every state that is relative $W[(S, P)]$-equivalent to $(S, P)$
- Extend this result to all states which belong to some run of $\mathcal{A}$, not just for the states in the class $\mathcal{C}([S_i, P_i])$
  - Fix an arbitrary initial state $(S, P)$ of $\mathcal{A}$ and define $\mathcal{M}$ as the *reflective* ASM machine with:

$$\mathcal{S}_{\mathcal{M}} = \{(S_i, P_i') \mid (S_i, P_i) \in \mathcal{S}_{\mathcal{A}} \text{ and } P_i' \text{ is the “self” representation of } r_{[(S_i,P_i)]}\}$$
$$\mathcal{I}_{\mathcal{M}} = \{(S_i, P_i') \mid (S_i, P_i') \in \mathcal{S}_{\mathcal{M}} \text{ and } P_i' \text{ is the “self” representation of } r_{[(S,P)]}\}$$

- With this show that for every run $(S_0, P_0), (S_1, P_1), \ldots$ of $\mathcal{A}$ and corresponding run $(S_0', P_0'), (S_1', P_1'), \ldots$ of $\mathcal{M}$ with $S_0 = S_0'$, it holds that $\Delta_{st}(r_{[(S_i,P_i')]}, (S_i', P_i')) = \Delta_{st}(\mathcal{A}, (S_i, P_i))$

# 3 Foundations: Logic

- Formulae of the logic for determistic ASMs (Stärk / Nanchen):

$$
\begin{aligned}
\varphi, \psi \ ::= \ & s = t \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \to \psi \mid \forall x.\varphi \mid \exists x.\varphi \mid \\
& \mid \mathrm{def}(r) \mid \mathrm{upd}(r, f, \vec{s}, t) \mid [r]\varphi
\end{aligned}
$$

  - Here $\mathrm{upd}(r, f, \vec{s}, t)$ informally means that rule $r$ yields an update at location $(f, \mathrm{val}_S(\vec{s}))$ with new value $\mathrm{val}_S(t)$

- A proof system has been defined – the logic is complete

  - The logic is a definitional extension of first-order logic

- The logic does not cover non-determinism

- The logic does not cover synchronisation of parallel branches

## 3.1 Extension to Non-Deterministic ASMs

- Formulae of the logic for non-determistic ASMs:

$$\begin{aligned}\varphi, \psi ::= {} & s = t \mid s_a = t_a \mid \neg\varphi \mid \varphi \wedge \psi \mid \forall x(\varphi) \mid \forall \mathtt{x}(\varphi) \mid \forall M(\varphi) \\ & \mid \forall X(\varphi) \mid \forall \mathcal{X}(\varphi) \mid \forall \ddot{X}(\varphi) \mid \forall \ddot{\mathcal{X}}(\varphi) \mid \forall F(\varphi) \mid \forall G(\varphi) \\ & \mid \mathrm{upd}(r, X) \mid \mathrm{upm}(r, \ddot{X}) \mid M(s, t_a) \mid X(f, t, t_0) \\ & \mid \mathcal{X}(f, t, t_0, s) \mid \ddot{X}(f, t, t_0, t_a) \mid \ddot{\mathcal{X}}(f, t, t_0, t_a, s) \\ & \mid F(f, t, t_0, t_a, t', t'_0, t'_a, s) \mid G(f, t, t_0, t_a, t', t'_0, t'_a, s_a) \mid [X]\varphi\end{aligned}$$

  - $s$, $t$ and $t'$ denote terms in $\mathcal{T}_f$
  - $s_a$, $t_a$ and $t'_a$ denote terms in $\mathcal{T}_a$
  - $x \in \mathcal{X}_f$ and $\mathtt{x} \in \mathcal{X}_a$ denote first-order variables
  - $M$, $X$, $\mathcal{X}$, $\ddot{X}$, $\ddot{\mathcal{X}}$, $F$ and $G$ denote second-order variables
  - $r$ is an ASM rule
  - $f$ is a dynamic function symbol in $\Upsilon_f \cup \mathcal{F}_b$
  - $t_0$ and $t'_0$ denote terms in $\mathcal{T}_f$ or $\mathcal{T}_a$ depending on whether $f$ is in $\Upsilon_f$ or $\mathcal{F}_b$, respectively

# Informal Meaning

- $\mathrm{upd}(r, X)$ and $\mathrm{upm}(r, \ddot{X})$ respectively state that a finite update set represented by $X$ and a finite update multiset represented by $\ddot{X}$ are generated by a rule $r$

- $X(f, t, t_0)$ describes that an update $(f, t, t_0)$ belongs to the update set represented by $X$

- $\ddot{X}(f, t, t_0, t_a)$ describes that an update $(f, t, t_0)$ occurs at least once in the update multiset represented by $\ddot{X}$

- If $(f, t, t_0)$ occurs $n$-times in the update multiset represented by $\ddot{X}$, then there are $n$ distinct $a_1, \ldots, a_n \in B_a$ such that $(f, t, t_0, a_i) \in \ddot{X}$ for every $1 \leq i \leq n$ and $(f, t, t_0, a_j) \notin \ddot{X}$ for every $a_j$ other than $a_1, \ldots, a_n$

- $[X]\varphi$ expresses that $\varphi$ holds in the state resulting from executing the update set represented by $X$ on the current state

## Completeness

- The second-order variables $\mathcal{X}$ and $\ddot{\mathcal{X}}$ are used to keep track of the parallel branches that produce the update sets and multisets, respectively

- $M$ denotes binary second-order variables which are used to represent the finite multisets in the semantic interpretation of $\rho$-terms
- $F$ and $G$ to denote second-order variables which encode bijections between update multisets

A proof system for this logic has been developed

**Theorem.** The logic of non-deterministic ASMs is complete with respect to Henkin semantics for higher-order logics.

F. Ferrarotti, K.-D. Schewe, L. Tec, Q. Wang: A complete logic for Database Abstract State Machines1. Logic Journal of the IGPL vol. 25 (5): 700-740 (2017)

## 3.2 Extension: Concurrency

- Simple observation: concurrency can be mimicked by non-determinism: for each agent $a$ replace its rule $r$ by

**IF** ctl = idle **THEN CHOOSE** $r$ **OR** local($r$) $\|$ ctl := active **ENDIF**
**IF** ctl = active **THEN CHOOSE** skip **OR** final($r$) $\|$ ctl := idle **ENDIF**

- In an initial state the "control-state" location ctl is set to idle
  - If idle the agent executes either immediately its rule or executes a local version of it, i.e. all updates will be written to a local copy
- Otherwise the control-state becomes active
  - If active, the agent may either do nothing or finalise the execution by copying all updates to the shared locations and returning to an idle control state

F. Ferrarotti, K.-D. Schewe, L. Tec, Q. Wang: A unifying logic for non-deterministic, parallel and concurrent Abstract State Machines. Annals of Mathematics and Artificial Intelligence (2018), to appear

# 3.3 Extension: Reflection

- Reflection concerns rules $r$ in the logic, which only appear in formulae of the form $\text{upd}(r, X)$ and $\text{upm}(r, \ddot{X})$

  - In a non-reflective ASM the main rule is given as part of the specification and treated as extra-logical constant

  - In a reflective ASM the main rule is the value in a location such as $\mathit{self}$: we have $val_S(\mathit{self}) = r_S$

  - That is, the interpretation of the term $\mathit{self}$ in a state $S$ yields the rule that is to be applied in $S$

- As in a reflective ASM the main rule does not change within a single step, we have to take multiple steps into account

# Predicates for Multiple Steps

- Use two additional predicates r-upd and r-upm with the following informal meaning:
  - r-upd$(n, X)$ means that $n$ steps of the reflective ASM yield the update set $X$, where in each step the actual value of *self* is used
  - r-upm$(n, X)$ means that $n$ steps of the reflective ASM yield the update multiset $X$
- The proof theory for non-deterministic ASMs used in the completeness proof defines upd$(r, X)$ and upm$(r, \ddot{X})$ for sequence rules
- Inductively define axioms for r-upd and r-upm
  - Clearly, we have r-upd$(1, X) \leftrightarrow$ upd$(\mathit{self}, X)$ and r-upm$(1, \ddot{X}) \leftrightarrow$ upm$(\mathit{self}, \ddot{X})$

# Predicates for Multiple Steps (cont.)

$$\text{r-upd}(n+1, X) \leftrightarrow \big(\text{r-upd}(1, X) \wedge \neg\text{conUSet}(X)\big) \vee$$
$$\big(\exists Y_1 Y_2 (\text{r-upd}(1, Y_1) \wedge \text{conUSet}(Y_1) \wedge [Y_1]\text{r-upd}(n, Y_2) \wedge$$
$$\bigwedge_{f \in \mathcal{F}_{dyn}} \forall xy (X(f, x, y) \leftrightarrow ((Y_1(f, x, y) \wedge \forall z(\neg Y_2(f, x, z))) \vee Y_2(f, x, y))))\big)$$

$$\text{upm}(n+1, \ddot{X}) \leftrightarrow \Big(\text{r-upm}(1, \ddot{X}) \wedge$$
$$\forall X \Big( \bigwedge_{f \in \mathcal{F}_{dyn}} \forall x_1 x_2 (X(f, x_1, x_2) \leftrightarrow \exists \mathtt{x}_3 (\ddot{X}(f, x_1, x_2, \mathtt{x}_3))) \wedge \neg\text{conUSet}(X)\Big)\Big) \vee$$
$$\Big(\exists \ddot{Y}_1 \ddot{Y}_2 \Big(\text{r-upm}(1, \ddot{Y}_1) \wedge \forall Y_1 \Big( \bigwedge_{f \in \mathcal{F}_{dyn}} \forall x_1 x_2 (Y_1(f, x_1, x_2) \leftrightarrow$$
$$\exists \mathtt{x}_3 (\ddot{Y}_1(f, x_1, x_2, \mathtt{x}_3))) \wedge \text{conUSet}(Y_1) \wedge [Y_1]\text{r-upm}(n, \ddot{Y}_2)\Big) \wedge$$
$$\bigwedge_{f \in \mathcal{F}_{dyn}} \forall x_1 x_2 \mathtt{x}_3 \big(\ddot{X}(f, x_1, x_2, \mathtt{x}_3) \leftrightarrow (\ddot{Y}_2(f, x_1, x_2, \mathtt{x}_3) \vee$$
$$(\ddot{Y}_1(f, x_1, x_2, \mathtt{x}_3) \wedge \forall y_2 \mathtt{y}_3 (\neg \ddot{Y}_2(f, x_1, y_2, \mathtt{y}_3)))))\Big)\Big)$$

# 4 Outlook

## Open Issues

- The behavioural theory of distributed adaptive systems still needs to be written up in an integrated way
- Agents in the theory are assumed to be deterministic; extensions to capture non-determinism are open
  - In particular, the case of unbounded parallelism in combination with unbounded choice appears to be at least as challenging as the behavioural theory of parallel algorithms
- The completeness of the extended logic for concurrent reflective algorithms is open
- Further extend the theory towards probabilistic choice with arbitrary distribution (not just equal distribution)
- In all cases the logic needs to be extended by integration probabilistic logic concepts

# Hybrid Systems

- In hybrid systems the sequential time postulate should become a **continuous time postulate** turning runs into continuous functions from $\mathbb{R}$ to the set of states
  - Using the usual topology on $\mathbb{R}$, product topology, discrete topology the set of states can be easily turned into a topological space
  - Showing an equivalence to discrete runs (as before) with continuous functions as values should be possible
  - With this equivalence an extension to hybrid ASMs appears to be straightforward
- A crucial problem concerns conditions, under which a discretisation of an (observed) continuous function can be used as surrogate for the continuous function itself
- Concerning the logic it is crucial to integrate functional (such as derivatives), maybe be looking into higher-order categorical logic

# Complexity

- Specifications in (concurrent, reflective) ASMs may also be exploited for analysing and classifying complexity

- State of the art in complexity theory still refers to Turing machines

  - In descriptive complexity theory many proofs construct logical formulae describing the behaviour of a particular Turing machine, which could be simplified using ASMs and other rigorous methods

  - Complexity classes based on “alternating” Turing machines refer to parallelism

  - Alternating sequences of quantifiers in descriptive complexity are closely linked to the interaction of choice and unbounded parallelism

***Thank you***