Introduction to ASM: Fundamentals

Adaptable and flexible specification's technique

Modeling in the correct abstraction level

Natural and easy understandable semantics.

Material: See http://www.di.unipi.it/AsmBook/

Theoretical fundaments: ASM Theses

Abstract state machines as computation models

Turing Machines (RAM, part.rec. Fct,..) serve as computation model, e.g. fixing the notion of computable functions. In principle is possible to simulate every algorithmic solution with an appropriate TM.

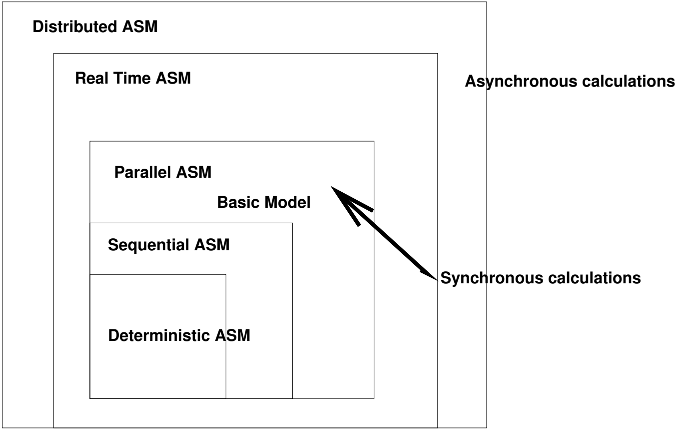
Problem: Simulation is not easy, because there are different abstraction levels of the manipulated objects and different granularity of the steps.

Question: Is it possible to generalize the TM in such a way that every algorithm, independent from it's abstraction level, can be naturally and faithfully simulated with such generalized machine?
How would the states and instructions of such a machine look like?

Easy: If Condition Then Action

ASM Thesis

ASM Thesis The concept of abstract state machine provides a universal computation model with the ability to simulate arbitrary algorithms on their natural levels of abstraction. Yuri Gurevich



Sequential ASM Thesis

- The model of the sequential ASM's is universal for all the sequential algorithms.
- Each sequential algorithm, independent from his abstraction level, can be simulated step by step by a sequential ASM.

To confirm this thesis we need definitions for sequential algorithms and for sequential ASM's.

$\rightsquigarrow$ Postulates for sequentiality

Sequentiality Postulates

- Sequential time:
  Computations are linearly arranged.
- Abstract states:
  Each kind of static mathematical reality can be represented by a structure of the first order logic (PL 1). (Tarski)
- Bounded exploration:
  Each computation step depends only on a finite (depending only on the algorithm) bounded state information.

Y. Gurevich:: Sequential Abstract State Machines Capture Sequential Algorithms, ACM Transactions on Computational Logic, 1, 2000, 77-111.

The postulates in detail: Sequential time

Let A be a sequential algorithm. To A belongs:

- A set (Set of states) $S(A)$ of States of A.
- A subset $I(A)$ of $S(A)$ which elements are called initial states of A.
- A mapping $\tau_A : S(A) \rightarrow S(A)$, the one-step-function of A.

An run (or a computation) of A is a finite or infinite sequence of states of A

$$X_0, X_1, X_2, \ldots$$

in which X_0 is an initial state and $\tau_A(X_i) = X_{i+1}$ holds for each i.

Logical time and not physical time.

Abstract States

Definition 3.1 (Equivalent algorithms). *Algorithms A and B are equivalent if* $S(A) = S(B)$, $I(A) = I(B)$ *and* $\tau_A = \tau_B$.
In particular equivalent algorithms have the same runs.

Let A be a sequential algorithm:

- States of A are first order (PL1) structures.
- All the states of A have the same vocabulary (signature).
- The one-step-function doesn't change the base set (universe) $B(X)$ of a state.
- $S(A)$ and $I(A)$ are closed under isomorphisms and each isomorphism from state X to state Y is also an isomorphism of state $\tau_A(X)$ to $\tau_A(Y)$.

Exercises

States: Signatures, interpretations, universe, terms, ground terms, value
...
Signatures (vocabulary): function- and relation-names, arity ($n \geq 0$)

Assumption: *true*, *false*, *undef* (constants), *Boole* (monadic) and $=$ are contained in every signature.
The interpretation of *true* is different from the one for *false*, *undef*.
Relations are considered as functions with the value of *true*, *false* in the interpretations.
Monadic relations are seen as subsets of the base set of the interpretations.
Let $Val(t, X)$ be the value in state X for a ground term t that is in the vocabulary.
Functions are divided in dynamic and static, according whether they can change or not, when a state transition occurs.
Exercise: Model the states of a TM as an abstract state.
Model the states of the standard Euclidean algorithm.

Bounded exploration

- Unbounded-Parallelism: Consider the following graph-reachability algorithm that iterates the following step. (It is assumed that at the beginning only one node satisfies the unary relation R.)

  do for all x, y with $Edge(x, y) \wedge R(x) \wedge \neg R(y)$ $\quad R(y) := true$

  In each computation step an unbounded number of local changes is made on a global state.

- Unbounded-Step-Information:
  Test for isolated nodes in a graph:

  if $\forall x \exists y$ $Edge(x, y)$ then Output := false else Output := true

  In one step only bounded local changes are made, though an unbounded part of the state is considered in one step.
  How can these properties be formalized? $\rightsquigarrow$ Atomic actions

Update sets

Consider the structure X as memory:

If f is a function name of arity j and $\overline{a}$ a j-tuple of base elements from X, then the pair $(f, \overline{a})$ is called a location and $Content_X(f, \overline{a})$ is the value of the interpretation of f for $\overline{a}$ in X.

Is $(f, \overline{a})$ a location of X and b an element of X, then $(f, \overline{a}, b)$ is called an update of X. The update is trivial when $b = Content_X(f, \overline{a})$.

To make (fire) an update, the actual content of the location is replaced by b.

A set of updates of X is consistent when in the set there is no pair of updates with the same location and different values.
A set Δ of updates is executed by making all updates in the set simultaneously (in case the set is consistent, in other case nothing is done).
The result is denoted by $X + \Delta$.

Update sets of algorithms, Reachable elements

Lemma 3.2. *If X, Y are structures over the same signature and with the same base set, then there is a unique consistent set Δ of non-trivial updates of X with $Y = X + \Delta$. Let $\Delta \leftrightharpoons Y - X$.*

Definition 3.3. *Let X be a state of algorithm A. According to the definition, X and $\tau_A(X)$ have the same signature and base set. Set:*

$$\Delta(A, X) \leftrightharpoons \tau_A(X) - X \quad \text{i.e. } \tau_A(X) = X + \Delta(A, X)$$

How can we bring up the elements of the base set in the description of the algorithm at all? $\rightsquigarrow$ Using the ground terms of the signature.

Definition 3.4 (Reachable element). *An element a of a structure X is reachable when $a = Val(t, X)$ for a ground term t in the vocabulary of X.*
A location $(f, \overline{a})$ of X is reachable when each element in the tuple $\overline{a}$ is reachable.
An update $(f, \overline{a}, b)$ of X is reachable when $(f, \overline{a})$ and b are reachable.

Bounded exploration postulate

Two structures X and Y with the same vocabulary Sig coincide on a set T of Sig- terms, when $Val(t, X) = Val(t, Y)$ for all $t \in T$. The vocabulary (signature) of an algorithm is the vocabulary of his states.

Let A be a sequential algorithm.

- There exist a finite set T of terms in the vocabulary of A, so that: $\Delta(A, X) = \Delta(A, Y)$, for all states X, Y of A, that coincide on T.

Intuition: Algorithm A examines only the part of a state that is reachable with the set of terms T. If two states coincide on this term-set, then the update-sets of the algorithm for both states should be the same.

The set T is a bounded-exploration witness for A.

Example

Example 3.5. *Consider algorithm A:*

$$\textit{if } P(f) \textit{ then } f := S(f)$$

States with interpretations with base set $\mathbb{N}$, *P subset of the natural numbers, for S the successor function and f a constant.*

Evidently A fulfills the postulates of sequential time and abstract states.

One could believe that
$T_0 = \{f, P(f), S(f)\}$ *is a bounded-exploration witness for A.*

Example: Continued

Let X be the canonical state of A with $f = 0$ and $P(0)$ holding.

Set $a \fallingdotseq Val(true, X)$ and $b \fallingdotseq Val(false, X)$, so that

$$Val(P(0), X) = Val(true, X) = a.$$

Let Y be the state that is obtained out of X through reinterpretation of *true* as b and *false* as a, i.e. $Val(true, Y) = b$ and $Val(false, Y) = a$. The values of f and $P(0)$ are left unchanged:

$Val(P(0), Y) = a$, thus $P(0)$ is not valid in Y.

Consequently X, Y coincide on T_0 but $\Delta(A, X) \neq \emptyset = \Delta(A, Y)$.

The set $T = T_0 \cup \{true\}$ is a bounded-exploration witness for A.

Sequential algorithms

Definition 3.6 (Sequential algorithm). *A sequential algorithm is an object A, which fulfills the three postulates.*
In particular A has a vocabulary and a bounded-exploration witness T. Without loss of generality (w.l.o.g.) T is subterm-closed and contains $true, false, undef$. The terms of T are called critical and their interpretations in a state X are called critical values in X.

Lemma 3.7. *If $(f, a_1, ..., a_j, a_0)$ is an update in $\Delta(A, X)$, then all the elements $a_0, a_1, ..., a_j$ are critical values in X.*

Proof: exercise (Proof by contradiction).
The set of the critical terms does not depend of X, thus there is a fixed upper bound for the size of $\Delta(A, X)$ and A changes in every step a bounded number of locations. Each one of the updates in $\Delta(A, X)$ is an atomic action of A. I.e. $\Delta(A, X)$ is a bounded set of atomic actions of A.

Sequential ASM-programs: Update rules

Definition 3.8 (Update rule). *An update rule over the signature Sig has the form*

$$f(t_1, ..., t_j) := t_0$$

in which f is a function and t_i are (ground) terms in Sig. To fire the rule in the Sig-structure X, compute the values $a_i = Val(t_i, X)$ and execute update $((f, a_1, ..., a_j), a_0)$ over X.
Parallel update rule over Sig: Let R_i be update rules over Sig, then

par
R_1
R_2
. *Notation: Block (when empty skip)*
.
.
R_k
endpar *fires through simultaneously firing of R_i.*

Sequential ASM-programs

Definition 3.9 (Semantics of update rules). *If R is an update rule $f(t_1, ..., t_j) := t_0$ and $a_i = Val(t_i, X)$ then set*

$$\Delta(R, X) \risingdotseq \{(f, (a_1, ..., a_j), a_0)\}$$

If R is a par-update rule with components $R_1, ...R_k$ then set

$$\Delta(R, X) \risingdotseq \Delta(R1, X) \cup \cdots \cup \Delta(Rk, X).$$

Consequence 3.10. *There exists in particular for each state X a rule R^X that uses only critical terms with* $\Delta(R^X, X) = \Delta(A, X)$.

Notice: If X, Y coincide on the critical terms, then $\Delta(R^X, Y) = \Delta(A, Y)$ holds. If X, Y are states and $\Delta(R^X, Z) = \Delta(A, Z)$ for a state Z, that is isomorphic to Y, then also $\Delta(R^X, Y) = \Delta(A, Y)$ holds.
Consider the equivalence relation $E_X(t1, t2) \risingdotseq Val(t1, X) = Val(t2, X)$ on T.
X, Y are T-similar, when $E_X = E_Y \rightsquigarrow \Delta(R^X, Y) = \Delta(A, Y)$. Exercise

Sequential ASM-programs

Definition 3.11. *Let φ be a boolean term over Sig (i.e. containing ground equations, not, and, or) and R_1, R_2 rules over Sig, then*

if φ then R_1
else R2
endif *is a rule*

Semantic:: To fire the rule in state X evaluate φ in X. If the result is true, then $\Delta(R, X) = \Delta(R_1, X)$, if not $\Delta(R, X) = \Delta(R_2, X)$.

Definition 3.12 (Sequential ASM program). *A sequential ASM program Π over the signature Sig is a rule over Sig. According to this $\Delta(\Pi, X)$ is well defined for each Sig-structure X. Let $\tau_\Pi(X) \leftrightharpoons X + \Delta(\Pi, X)$.*

Lemma 3.13. ***Basic result:*** *For each sequential algorithm A over Sig there's a sequential ASM-programm Π over Sig with $\Delta(\Pi, X) = \Delta(A, X)$ for all the states X of A.*

Sequential ASM-machines

Definition 3.14 (A sequential abstract-state-machine (seq-ASM)). *A seq-ASM B over the signature Σ is given through:*

- *A sequential ASM-programm Π over Σ.*
- *A set $S(B)$ of interpretations of Σ that is closed under isomorphisms and under the mapping τ_Π .*
- *A subset $I(B) \subset S(B)$, that is closed under isomorphisms.*

Theorem 3.15. *For each sequential algorithm A there is an equivalent sequential ASM.*

Example

Example 3.16. *Maximal interval-sum.[Gries 1990]. Let A be a function from* $\{0, 1, ..., n-1\} \to \mathbb{R}$ *and* $i, j, k \in \{0, 1, ..., n\}$.
For $i \leq j$: $S(i,j) \rightleftharpoons \sum_{i \leq k < j} A(k)$. *In particular* $S(i,i) = 0$.

Problem: *Compute* $S \rightleftharpoons max_{i \leq j} S(i,j)$.

Define $y(k) \rightleftharpoons max_{i \leq j \leq k} S(i,j)$. Then $y(0) = 0, y(n) = S$ and

$$y(k+1) = max\{max_{i \leq j \leq k} S(i,j), max_{i \leq k+1} S(i, k+1)\} = max\{y(k), x(k+1)\}$$

where $x(k) \rightleftharpoons max_{i \leq k} S(i,k)$, thus $x(0) = 0$ and

$$\begin{aligned} x(k+1) &= max\{max_{i \leq k} S(i, k+1), S(k+1, k+1)\} \\ &= max\{max_{i \leq k}(S(i,k) + A(k)), 0\} \\ &= max\{(max_{i \leq k} S(i,k)) + A(k), 0\} \\ &= max\{x(k) + A(k), 0\} \end{aligned}$$

Continuation of the example

Due to $y(k) \geq 0$, we have

$$y(k+1) = max\{y(k), x(k+1)\} = max\{y(k), x(k) + A(k)\}$$

Assumption: The 0-ary dynamic functions k, x, y are 0 in the initial state. The required algorithm is then

$$
\begin{array}{l}
\mathit{if} \quad k \neq n \quad \mathit{then} \\
\qquad \mathit{par} \\
\qquad\quad x := max\{x + A(k), 0\} \\
\qquad\quad y := max\{y, x + A(k)\} \\
\qquad\quad k := k + 1 \\
\mathit{else} \quad S := y
\end{array}
$$

Exercise 3.17. *Simulation*
Define an ASM, that implements Markov's Normal-algorithms.
e.g. for $ab \rightarrow A,\ ba \rightarrow B,\ c \rightarrow C$

Detailed definition of ASMs

- Part 1: Abstract states and update sets
- Part 2: Mathematical Logic
- Part 3: Transition rules and runs of ASMs
- Part 4: The reserve of ASMs

 1

Part 1

Abstract states and update sets

 2

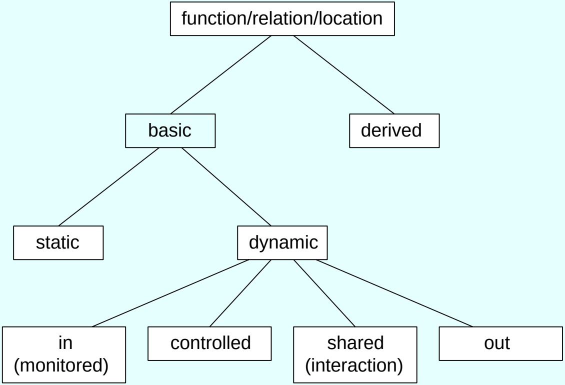
Signatures

Definition. A *signature* Σ is a finite collection of function names.

- Each function name f has an *arity*, a non-negative integer.
- Nullary function names are called *constants*.
- Function names can be *static* or *dynamic*.
- Every ASM signature contains the static constants $undef$, $true$, $false$.

Signatures are also called *vocabularies*.

Classification of functions



 4

States

Definition. A *state* $\mathfrak{A}$ for the signature Σ is a non-empty set X, the *superuniverse* of $\mathfrak{A}$, together with an *interpretation* $f^{\mathfrak{A}}$ of each function name f of Σ.

- If f is an n-ary function name of Σ, then $f^{\mathfrak{A}}: X^n \to X$.
- If c is a constant of Σ, then $c^{\mathfrak{A}} \in X$.
- The superuniverse X of the state $\mathfrak{A}$ is denoted by $|\mathfrak{A}|$.

- The superuniverse is also called the *base set* of the state.
- The *elements* of a state are the elements of the superuniverse.

 5

States (continued)

- The interpretations of $undef$, $true$, $false$ are pairwise different.
- The constant $undef$ represents an undetermined object.
- The *domain* of an n-ary function name f in $\mathfrak{A}$ is the set of all n-tuples $(a_1, \ldots, a_n) \in |\mathfrak{A}|^n$ such that $f^{\mathfrak{A}}(a_1, \ldots, a_n) \neq undef^{\mathfrak{A}}$.
- A *relation* is a function that has the values $true$, $false$ or $undef$.
- We write $a \in R$ as an abbreviation for $R(a) = true$.
- The superuniverse can be divided into *subuniverses* represented by unary relations.

Locations

Definition. A *location* of $\mathfrak{A}$ is a pair

$$(f, (a_1, \ldots, a_n))$$

where f is an n-ary function name and $a_1, \ldots, a_n$ are elements of $\mathfrak{A}$.

- The value $f^{\mathfrak{A}}(a_1, \ldots, a_n)$ is the *content* of the location in $\mathfrak{A}$.
- The *elements* of the location are the elements of the set $\{a_1, \ldots, a_n\}$.
- We write $\mathfrak{A}(l)$ for the content of the location l in $\mathfrak{A}$.

Notation. If $l = (f, (a_1, \ldots, a_n))$ is a location of $\mathfrak{A}$ and α is a function defined on $|\mathfrak{A}|$, then $\alpha(l) = (f, (\alpha(a_1), \ldots, \alpha(a_n)))$.

 7

Updates and update sets

Definition. An *update* for $\mathfrak{A}$ is a pair (l, v), where l is a location of $\mathfrak{A}$ and v is an element of $\mathfrak{A}$.

- The update is *trivial*, if $v = \mathfrak{A}(l)$.
- An *update set* is a set of updates.

Definition. An update set U is *consistent*, if it has no clashing updates, i.e., if for any location l and all elements v, w, if $(l, v) \in U$ and $(l, w) \in U$, then $v = w$.

Firing of updates

Definition. The result of *firing* a consistent update set U in a state $\mathfrak{A}$ is a new state $\mathfrak{A} + U$ with the same superuniverse as $\mathfrak{A}$ such that for every location l of $\mathfrak{A}$:

$$(\mathfrak{A} + U)(l) = \begin{cases} v, & \text{if } (l, v) \in U; \\ \mathfrak{A}(l), & \text{if there is no } v \text{ with } (l, v) \in U. \end{cases}$$

The state $\mathfrak{A} + U$ is called the *sequel* of $\mathfrak{A}$ with respect to U.

 9

Homomorphisms and isomorphisms

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two states over the same signature.

Definition. A *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a function α from $|\mathfrak{A}|$ into $|\mathfrak{B}|$ such that $\alpha(\mathfrak{A}(l)) = \mathfrak{B}(\alpha(l))$ for each location l of $\mathfrak{A}$.

Definition. An *isomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ which is a ono-to-one function from $|\mathfrak{A}|$ onto $|\mathfrak{B}|$.

Lemma (Isomorphism). Let α be an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. If U is a consistent update set for $\mathfrak{A}$, then $\alpha(U)$ is a consistent update set for $\mathfrak{B}$ and α is an isomorphism from $\mathfrak{A}+U$ to $\mathfrak{B}+\alpha(U)$.

Composition of update sets

$$U \oplus V = V \cup \{(l, v) \in U \mid \text{there is no } w \text{ with } (l, w) \in V\}$$

Lemma. Let U, V, W be update sets.

- $(U \oplus V) \oplus W = U \oplus (V \oplus W)$
- If U and V are consistent, then $U \oplus V$ is consistent.
- If U and V are consistent, then $\mathfrak{A} + (U \oplus V) = (\mathfrak{A} + U) + V$.

Part 2

Mathematical Logic

 12

Terms

Let Σ be a signature.

Definition. The *terms* of Σ are syntactic expressions generated as follows:

- Variables x, y, z, ... are terms.
- Constants c of Σ are terms.
- If f is an n-ary function name of Σ, $n > 0$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

- A term which does not contain variables is called a *ground term*.
- A term is called *static*, if it contains static function names only.
- By $t\frac{s}{x}$ we denote the result of replacing the variable x in term t everywhere by the term s (*substitution* of s for x in t).

Variable assignments

Let $\mathfrak{A}$ be a state.

Definition. A *variable assignment* for $\mathfrak{A}$ is a finite function ζ which assigns elements of $|\mathfrak{A}|$ to a finite number of variables.

- We write $\zeta[x \mapsto a]$ for the variable assignment which coincides with ζ except that it assigns the element a to the variable x:

$$\zeta[x \mapsto a](y) = \begin{cases} a, & \text{if } y = x; \\ \zeta(y), & \text{otherwise.} \end{cases}$$

- Variable assignments are also called *environments*.

Evaluation of terms

Definition. Let $\mathfrak{A}$ be a state of Σ.
Let ζ be a variable assignment for $\mathfrak{A}$.
Let t be a term of Σ such that all variables of t are defined in ζ.
The *value* $[\![t]\!]_\zeta^\mathfrak{A}$ is defined as follows:

- $[\![x]\!]_\zeta^\mathfrak{A} = \zeta(x)$
- $[\![c]\!]_\zeta^\mathfrak{A} = c^\mathfrak{A}$
- $[\![f(t_1, \ldots, t_n)]\!]_\zeta^\mathfrak{A} = f^\mathfrak{A}([\![t_1]\!]_\zeta^\mathfrak{A}, \ldots, [\![t_n]\!]_\zeta^\mathfrak{A})$

Evaluation of terms (continued)

Lemma (Coincidence). If ζ and η are two variable assignments for t such that $\zeta(x) = \eta(x)$ for all variables x of t, then $[\![t]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\eta}$.

Lemma (Homomorphism). If α is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$, then $\alpha([\![t]\!]^{\mathfrak{A}}_{\zeta}) = [\![t]\!]^{\mathfrak{B}}_{\alpha \circ \zeta}$ for each term t.

Lemma (Substitution). Let $a = [\![s]\!]^{\mathfrak{A}}_{\zeta}$.
Then $[\![t\frac{s}{x}]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\zeta[x \mapsto a]}$.

Formulas

Let Σ be a signature.

Definition. The *formulas* of Σ are generated as follows:

- If s and t are terms of Σ, then $s = t$ is a formula.
- If φ is a formula, then $\neg\varphi$ is a formula.
- If φ and ψ are formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \to \psi)$ are formulas.
- If φ is a formula and x a variable, then $(\forall x\, \varphi)$ and $(\exists x\, \varphi)$ are formulas.

- A formula $s = t$ is called an *equation*.
- The expression $s \neq t$ is an abbreviation for $\neg(s = t)$.

 17

Formulas (continued)

symbol	name	meaning
$\neg$	negation	not
$\wedge$	conjunction	and
$\vee$	disjunction	or (inclusive)
$\rightarrow$	implication	if-then
$\forall$	universal quantification	for all
$\exists$	existential quantification	there is

 18

Formulas (continued)

$$\varphi \wedge \psi \wedge \chi \quad \text{stands for} \quad ((\varphi \wedge \psi) \wedge \chi),$$
$$\varphi \vee \psi \vee \chi \quad \text{stands for} \quad ((\varphi \vee \psi) \vee \chi),$$
$$\varphi \wedge \psi \to \chi \quad \text{stands for} \quad ((\varphi \wedge \psi) \to \chi), \text{ etc.}$$

- The variable x is *bound* by the quantifier $\forall$ ($\exists$) in $\forall x\, \varphi$ ($\exists x\, \varphi$).
- The *scope* of x in $\forall x\, \varphi$ ($\exists x\, \varphi$) is the formula φ.
- A variable x occurs *free* in a formula, if it is not in the scope of a quantifier $\forall x$ or $\exists x$.
- By $\varphi\frac{t}{x}$ we denote the result of replacing all free occurrences of the variable x in φ by the term t. (Bound variables are renamed.)

Semantics of formulas

$$[\![s = t]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![s]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\zeta}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\neg\varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{false}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \wedge \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true} \text{ and } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \vee \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true} \text{ or } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \rightarrow \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{false} \text{ or } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\forall x\, \varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x\mapsto a]} = \mathit{true} \text{ for every } a \in |\mathfrak{A}|; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\exists x\, \varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} \mathit{true}, & \text{if there exists an } a \in |\mathfrak{A}| \text{ with } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x\mapsto a]} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

Coincidence, Substitution, Isomorphism

Lemma (Coincidence). If ζ and η are two variable assignments for φ such that $\zeta(x) = \eta(x)$ for all free variables x of φ, then $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{A}}_{\eta}$.

Lemma (Substitution). Let t be a term and $a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$. Then $[\![\varphi\frac{t}{x}]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x\mapsto a]}$.

Lemma (Isomorphism). Let α be an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Then $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{B}}_{\alpha\circ\zeta}$.

 21

Models

Definition. A state $\mathfrak{A}$ is a *model* of φ (written $\mathfrak{A} \models \varphi$), if $[\![\varphi]\!]_{\zeta}^{\mathfrak{A}} = true$ for all variable assignments ζ for φ.

22

Part 3

Transition rules and runs of ASMs

 23

Transition rules

Skip Rule: **skip**

Meaning: Do nothing

Update Rule: $f(s_1, \ldots, s_n) := t$

Meaning: Update the value of f at $(s_1, \ldots, s_n)$ to t.

Block Rule: P **par** Q

Meaning: P and Q are executed in parallel.

Conditional Rule: **if** φ **then** P **else** Q

Meaning: If φ is true, then execute P, otherwise execute Q.

Let Rule: **let** $x = t$ **in** P

Meaning: Assign the value of t to x and then execute P.

 24

Transition rules (continued)

Forall Rule: **forall** x **with** φ **do** P

Meaning: Execute P in parallel for each x satisfying φ.

Choose Rule: **choose** x **with** φ **do** P

Meaning: Choose an x satisfying φ and then execute P.

Sequence Rule: P **seq** Q

Meaning: P and Q are executed sequentially, first P and then Q.

Call Rule: $r(t_1, \ldots, t_n)$

Meaning: Call transition rule r with parameters $t_1, \ldots, t_n$.

 25

Variations of the syntax

if φ **then**
P
else
Q
endif
	if φ **then** P **else** Q

[**do in-parallel**] P_1 ⋮ P_n [**enddo**]	P_1 **par** ... **par** P_n
$\{P_1, \ldots, P_n\}$	P_1 **par** ... **par** P_n

Variations of the syntax (continued)

do forall $x\colon\varphi$ P **enddo**	**forall** x **with** φ **do** P
choose $x\colon\varphi$ P **endchoose**	**choose** x **with** φ **do** P
step P **step** Q	P **seq** Q

 27

Example

Example 3.18. *Sorting of linear data structures in-place, one-swap-a-time.*
Let $a : Index \to Value$

$$\begin{array}{l} choose \quad x, y \in Index : x < y \wedge a(x) > a(y) \\ \quad do \quad in - parallel \\ \quad\quad a(x) := a(y) \\ \quad\quad a(y) := a(x) \end{array}$$

Two kinds of non-determinisms:

"Don't-care" non-determinism: random choice

$$\begin{array}{l} choose \quad x \in \{x_1, x_2, ..., x_n\} \; with \quad \varphi(x) \quad do \\ \quad\quad R(x) \end{array}$$

"Don't-know" indeterminism

Extern controlled actions and events (e.g. input actions)

$$monitored \quad f : X \to Y$$

Free and bound variables

Definition. An occurrence of a variable x is *free* in a transition rule, if it is not in the scope of a **let** x, **forall** x or **choose** x.

$$\textbf{let } x = t \underbrace{\textbf{ in } P}_{\text{scope of } x}$$

$$\textbf{forall } x \underbrace{\textbf{ with } \varphi \textbf{ do } P}_{\text{scope of } x}$$

$$\textbf{choose } x \underbrace{\textbf{ with } \varphi \textbf{ do } P}_{\text{scope of } x}$$

 28

Rule declarations

Definition. A *rule declaration* for a rule name r of arity n is an expression

$$r(x_1, \ldots, x_n) = P$$

where

- P is a transition rule and
- the free variables of P are contained in the list $x_1, \ldots, x_n$.

Remark: Recursive rule declarations are allowed.

Abstract State Machines

Definition. An *abstract state machine* M consists of

- a signature Σ,
- a set of initial states for Σ,
- a set of rule declarations,
- a distinguished rule name of arity zero called the *main rule name* of the machine.

 30

Semantics of transition rules

The semantics of transition rules is defined in a calculus by rules:

$$\frac{\mathit{Premise}_1 \quad \cdots \quad \mathit{Premise}_n}{\mathit{Conclusion}}\ \mathit{Condition}$$

The predicate

$$\text{yields}(P, \mathfrak{A}, \zeta, U)$$

means:

The transition rule P yields the update set U in state $\mathfrak{A}$ under the variable assignment ζ.

Semantics of transition rules (continued)

$$\overline{\mathsf{yields}(\mathbf{skip}, \mathfrak{A}, \zeta, \emptyset)}$$

$$\overline{\mathsf{yields}(f(s_1, \dots, s_n) := t, \mathfrak{A}, \zeta, \{(l, v)\})}$$ where $l = (f, ([\![s_1]\!]^{\mathfrak{A}}_{\zeta}, \dots, [\![s_n]\!]^{\mathfrak{A}}_{\zeta}))$ and $v = [\![t]\!]^{\mathfrak{A}}_{\zeta}$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta, U) \quad \mathsf{yields}(Q, \mathfrak{A}, \zeta, V)}{\mathsf{yields}(P \ \mathbf{par} \ Q, \mathfrak{A}, \zeta, U \cup V)}$$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta, U)}{\mathsf{yields}(\mathbf{if} \ \varphi \ \mathbf{then} \ P \ \mathbf{else} \ Q, \mathfrak{A}, \zeta, U)}$$ if $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true}$

$$\frac{\mathsf{yields}(Q, \mathfrak{A}, \zeta, V)}{\mathsf{yields}(\mathbf{if} \ \varphi \ \mathbf{then} \ P \ \mathbf{else} \ Q, \mathfrak{A}, \zeta, V)}$$ if $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{false}$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\mathsf{yields}(\mathbf{let} \ x = t \ \mathbf{in} \ P, \mathfrak{A}, \zeta, U)}$$ where $a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U_a) \quad \text{for each } a \in I}{\mathsf{yields}(\mathbf{forall} \ x \ \mathbf{with} \ \varphi \ \mathbf{do} \ P, \mathfrak{A}, \zeta, \bigcup_{a \in I} U_a)}$$ where $I = \mathit{range}(x, \varphi, \mathfrak{A}, \zeta)$

 32

Semantics of transition rules (continued)

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\text{yields}(\textbf{choose } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, U)} \quad \text{if } a \in range(x, \varphi, \mathfrak{A}, \zeta)$$

$$\frac{}{\text{yields}(\textbf{choose } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, \emptyset)} \quad \text{if } range(x, \varphi, \mathfrak{A}, \zeta) = \emptyset$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A} + U, \zeta, V)}{\text{yields}(P \textbf{ seq } Q, \mathfrak{A}, \zeta, U \oplus V)} \quad \text{if } U \text{ is consistent}$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U)}{\text{yields}(P \textbf{ seq } Q, \mathfrak{A}, \zeta, U)} \quad \text{if } U \text{ is inconsistent}$$

$$\frac{\text{yields}(P\frac{t_1 \cdots t_n}{x_1 \cdots x_n}, \mathfrak{A}, \zeta, U)}{\text{yields}(r(t_1, \ldots, t_n), \mathfrak{A}, \zeta, U)} \quad \text{where } r(x_1, \ldots, x_n) = P \text{ is a rule declaration of } M$$

$$range(x, \varphi, \mathfrak{A}, \zeta) = \{a \in |\mathfrak{A}| : [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x \mapsto a]} = true\}$$

 33

Coincidence, Substitution, Isomorphisms

Lemma (Coincidence). If $\zeta(x) = \eta(x)$ for all free variables x of a transition rule P and P yields U in $\mathfrak{A}$ under ζ, then P yields U in $\mathfrak{A}$ under η.

Lemma (Substitution). Let t be a static term and $a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$. Then the rule $P\frac{t}{x}$ yields the update set U in state $\mathfrak{A}$ under ζ iff P yields U in $\mathfrak{A}$ under $\zeta[x \mapsto a]$.

Lemma (Isomorphism). If α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ and P yields U in $\mathfrak{A}$ under ζ, then P yields $\alpha(U)$ in $\mathfrak{B}$ under $\alpha \circ \zeta$.

Move of an ASM

Definition. A machine M can make a *move* from state $\mathfrak{A}$ to $\mathfrak{B}$ (written $\mathfrak{A} \stackrel{M}{\Longrightarrow} \mathfrak{B}$), if the main rule of M yields a consistent update set U in state $\mathfrak{A}$ and $\mathfrak{B} = \mathfrak{A} + U$.

- The updates in U are called *internal updates*.
- $\mathfrak{B}$ is called the *next internal state*.

If α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{A}'$, the following diagram commutes:

$$\begin{array}{ccc} \mathfrak{A} & \stackrel{M}{\Longrightarrow} & \mathfrak{B} \\ \alpha \downarrow & & \downarrow \alpha \\ \mathfrak{A}' & \stackrel{M}{\Longrightarrow} & \mathfrak{B}' \end{array}$$

Run of an ASM

Let M be an ASM with signature Σ.

A *run* of M is a finite or infinite sequence $\mathfrak{A}_0, \mathfrak{A}_1, \ldots$ of states for Σ such that

- $\mathfrak{A}_0$ is an initial state of M
- for each n,
  - either M can make a move from $\mathfrak{A}_n$ into the next internal state $\mathfrak{A}'_n$ and the environment produces a consistent set of external or shared updates U such that $\mathfrak{A}_{n+1} = \mathfrak{A}'_n + U$,
  - or M cannot make a move in state $\mathfrak{A}_n$ and $\mathfrak{A}_n$ is the last state in the run.

- In *internal* runs, the environment makes no moves.
- In *interactive* runs, the environment produces updates.

 36

Example

Example 3.19. *Minimal spanning tree:: **Prim's algorithm***

Two separated phases: initial, run

Signature: Weighted graph (connected, without loops) given by sets
NODE, EDGE, ... functions
$weight : EDGE \to REAL, frontier : EDGE \to Bool, tree : EDGE \to Bool$

$$
\begin{array}{l}
\textit{if} \quad mode = initial \quad \textit{then} \\
\quad \textit{choose} \quad p : NODE \\
\quad\quad Selected(p) := true \\
\quad\quad \textit{forall} \quad e : EDGE : p \in endpoints(e) \\
\quad\quad\quad frontier(e) := true \\
\quad mode := run
\end{array}
$$

Example: Prim's algorithm (Cont.)

$$
\begin{array}{l}
\textit{if} \quad mode = run \quad \textit{then} \\
\quad \textit{choose} \quad e : EDGE : frontier(e) \wedge \\
\qquad ((\forall f \in EDGE) : \; frontier(f) \Rightarrow \; weight(f) \geq weight(e)) \\
\quad\quad tree(e) := true \\
\quad\quad \textit{choose} \quad p : \; NODE : p \in endpoints(e) \wedge \neg Selected(p) \\
\quad\quad\quad Selected(p) := true \\
\quad\quad\quad \textit{forall} \quad f : EDGE : p \in endpoints(f) \\
\quad\quad\quad\quad frontier(f) := \neg frontier(f) \\
\quad \textit{ifnone} \quad mode := done
\end{array}
$$

How can we prove the correctness, termination?

Exercise 3.20. *Construct an ASM-Machine that implements Kruskal's algorithm.*

Part 4

The reserve of ASMs

 37

Importing new elements from the reserve

Import rule: **import** x **do** P

Meaning: Choose an element x from the reserve, delete it from the reserve and execute P.

let $x = new(X)$ **in** P abbreviates

import x **do**
  $X(x) := true$
  P

The reserve of a state

- New dynamic relation $Reserve$.
- $Reserve$ is updated by the system, not by rules.
- $Res(\mathfrak{A}) = \{a \in |\mathfrak{A}| : Reserve^{\mathfrak{A}}(a) = true\}$
- The reserve elements of a state are not allowed to be in the domain and range of any basic function of the state.

Definition. A state $\mathfrak{A}$ satisfies the *reserve condition* with respect to an environment ζ, if the following two conditions hold for each element $a \in Res(\mathfrak{A}) \setminus ran(\zeta)$:

- The element a is not the content of a location of $\mathfrak{A}$.
- If a is an element of a location l of $\mathfrak{A}$ which is not a location for $Reserve$, then the content of l in $\mathfrak{A}$ is $undef$.

Semantics of ASMs with a reserve

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\text{yields}(\textbf{import } x \textbf{ do } P, \mathfrak{A}, \zeta, V)} \quad \text{if } a \in Res(\mathfrak{A}) \setminus ran(\zeta) \text{ and } V = U \cup \{((Reserve, a), false)\}$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A}, \zeta, V)}{\text{yields}(P \textbf{ par } Q, \mathfrak{A}, \zeta, U \cup V)} \quad \text{if } Res(\mathfrak{A}) \cap El(U) \cap El(V) \subseteq ran(\zeta)$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U_a) \quad \text{for each } a \in I}{\text{yields}(\textbf{forall } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, \bigcup_{a \in I} U_a)} \quad \text{if } I = range(x, \varphi, \mathfrak{A}, \zeta) \text{ and for } a \neq b \; Res(\mathfrak{A}) \cap El(U_a) \cap El(U_b) \subseteq ran(\zeta)$$

- $El(U)$ is the set of elements that occur in the updates of U.
- The elements of an update (l, v) are the value v and the elements of the location l.

 40

Problem

Problem 1: New elements that are imported in parallel must be different.

import x **do** $parent(x) = root$
import y **do** $parent(y) = root$

Problem 2: Hiding of bound variables.

import x **do**
  $f(x) := 0$
  let $x = 1$ **in**
    import y **do** $f(y) := x$

Syntactic constraint. In the scope of a bound variable the same variable should not be used again as a bound variable (**let**, **forall**, **choose**, **import**).

Preservation of the reserve condition

Lemma (Preservation of the reserve condition).
If a state $\mathfrak{A}$ satisfies the reserve condition wrt. ζ and P yields a consistent update set U in $\mathfrak{A}$ under ζ, then

- the sequel $\mathfrak{A} + U$ satisfies the reserve condition wrt. ζ,
- $Res(\mathfrak{A} + U) \setminus ran(\zeta)$ is contained in $Res(\mathfrak{A}) \setminus El(U)$.

Permutation of the reserve

Lemma (Permutation of the reserve). Let $\mathfrak{A}$ be a state that satisfies the reserve condition wrt. ζ. If α is a function from $|\mathfrak{A}|$ to $|\mathfrak{A}|$ that permutes the elements in $Res(\mathfrak{A}) \setminus ran(\zeta)$ and is the identity on non-reserve elements of $\mathfrak{A}$ and on elements in the range of ζ, then α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{A}$.

 43

Independence of the choice of reserve elements

Lemma (Independence).
Let P be a rule of an ASM without **choose**. If

- $\mathfrak{A}$ satisfies the reserve condition wrt. ζ,
- the bound variables of P are not in the domain of ζ,
- P yields U in $\mathfrak{A}$ under ζ,
- P yields U' in $\mathfrak{A}$ under ζ,

then there exists a permutation α of $Res(\mathfrak{A}) \setminus ran(\zeta)$ such that $\alpha(U) = U'$.

Example: Abstract Data Types (ADT)

Example 3.21. *Double-linked lists*

See ASM-Buch.

Exercise 3.22. *Give an ASM-Specification for the data structure bounded stack.*

Distributed ASM: Concurrency, reactivity, time

Distributed ASM (DASM)

- Computation model:
  - Asynchronous computations
  - Autonomous operating agents
- A finite set of autonomous ASM-agents, each with a program of his own.
- Agents interact through reading and writing common locations of global machine states.
- Potential conflicts are solved through the underlying semantic model, according to the definition of (partial-ordered) runs.

Foundations: Orders, CPO's, Proof techniques

Properties of binary relations

- X set
- $\rho \subseteq X \times X$ binary relation
- Properties

(P1)	$x\,\rho\,x$	(reflexive)
(P2)	$(x\,\rho\,y \wedge y\,\rho\,x) \rightarrow x = y$	(antisymmetric)
(P3)	$(x\,\rho\,y \wedge y\,\rho\,z) \rightarrow x\,\rho\,z$	(transitive)
(P4)	$(x\,\rho\,y \vee y\,\rho\,x)$	(linear)

Quasi-Orders

- $\lesssim \subseteq X \times X$ Quasi-order iff $\lesssim$ reflexive and transitive.
- Kernel:
$$\approx \;=\; \lesssim \cap \lesssim^{-1}$$
- Strict part: $< \;=\; \lesssim \setminus \approx$
- $Y \subseteq X$ left-closed (in respect of $\lesssim$) iff
$$(\forall y \in Y : (\forall x \in X : x \lesssim y \rightarrow x \in Y))$$
- Notation: Quasi-order $(X, \lesssim)$

Partial-Orders

- $\leq \subseteq X \times X$ partial-order iff $\leq$ reflexive, antisymmetric and transitive.
- Kernel: Following holds

$$\mathrm{id}_X = \leq \cap \leq^{-1}$$

- Strict part: $< \;=\; \leq \setminus \mathrm{id}_X$
- Often: $<$ Partial-order iff $<$ irreflexive, transitive.
- Notation: Partial-order $(X, \leq)$

Well-founded Orderings

- Partial-order $\leq \subseteq X \times X$ well-founded iff

$$(\forall Y \subseteq X : Y \neq \emptyset \rightarrow (\exists y \in Y : y \text{ minimal in } Y \text{ in respect of } \leq))$$

- Quasi-order $\lesssim$ well-founded iff strict part of $\lesssim$ is well-founded.
- Initial segment: $Y \subseteq X$, left-closed
- Initial section of x: $\text{sec}(x) = \{y : y < x\}$

Supremum

- Let $(X, \leq)$ be a partial-order and $Y \subseteq X$
- $S \subseteq X$ is a chain iff elements of S are linearly ordered through $\leq$.
- y is an upper bound of Y iff

$$\forall y' \in Y : y' \leq y$$

- Supremum: y is a supremum of Y iff y is an upper bound of Y and

$$\forall y' \in X : ((y' \text{ upper bound of } Y) \rightarrow y \leq y')$$

- Analog: lower bound, Infimum $\inf(Y)$

CPO

- A Partial-order $(D, \sqsubseteq)$ is a complete partial ordering (CPO) iff
  - $\exists$ the smallest element $\perp$ of D (with respect of $\sqsubseteq$)
  - Each chain S has a supremum $\sup(S)$.

Example

Example 4.1.

- $(\mathcal{P}(X), \subseteq)$ *is CPO.*
- $(D, \sqsubseteq)$ *is CPO with*
  - $D = X \nrightarrow Y$: *set of all the partial functions* f *with* $\text{dom}(f) \subseteq X$ *and* $\text{cod}(f) \subseteq Y$.
  - *Let* $f, g \in X \nrightarrow Y$.

$$f \sqsubseteq g \textit{ iff } \text{dom}(f) \subseteq \text{dom}(g) \land (\forall x \in \text{dom}(f) : f(x) = g(x))$$

Monotonous, continuous

- $(D, \sqsubseteq)$, $(E, \sqsubseteq')$ CPOs
- $f : D \rightarrow E$ monotonous iff

$$(\forall d, d' \in D : d \sqsubseteq d' \rightarrow f(d) \sqsubseteq' f(d'))$$

- $f : D \rightarrow E$ continuous iff f monotonous and

$$(\forall S \subseteq D : S \text{ chain } \rightarrow f(\sup(S)) = \sup(f(S)))$$

- $X \subseteq D$ is admissible iff

$$(\forall S \subseteq X : S \text{ chain } \rightarrow \sup(S) \in X)$$

Fixpoint

- $(D, \sqsubseteq)$ CPO, $f : D \to D$
- $d \in D$ fixpoint of f iff

$$f(d) = d$$

- $d \in D$ smallest fixpoint of f iff d fixpoint of f and

$$(\forall d' \in D : d' \text{ fixpoint } \to d \sqsubseteq d')$$

Fixpoint-Theorem

Theorem 4.2 (Fixpoint-Theorem:). *$(D, \sqsubseteq)$ CPO, $f : D \to D$ continuous, then f has a smallest fixpoint μf and*

$$\mu f = \sup\{f^i(\bot) : i \in \mathbb{N}\}$$

Proof: (Sketch)

- $\sup\{f^i(\bot) : i \in \mathbb{N}\}$ fixpoint:

$$\begin{aligned} f(\sup\{f^i(\bot) : i \in \mathbb{N}\}) &= \sup\{f^{i+1}(\bot) : i \in \mathbb{N}\} \\ & \quad \text{(continuous)} \\ &= \sup\{\sup\{f^{i+1}(\bot) : i \in \mathbb{N}\}, \bot\} \\ &= \sup\{f^i(\bot) : i \in \mathbb{N}\} \end{aligned}$$

Fixpoint-Theorem (Cont.)

Fixpoint-Theorem: $(D, \sqsubseteq)$ CPO, $f : D \rightarrow D$ continuous, then f has a smallest fixpoint μf and

$$\mu f = \sup\{f^i(\bot) : i \in \mathbb{N}\}$$

Proof: (Continuation)

- $\sup\{f^i(\bot) : i \in \mathbb{N}\}$ smallest fixpoint:
  1. d' fixpoint of f
  2. $\bot \sqsubseteq d'$
  3. f monotonous, d' FP: $f(\bot) \sqsubseteq f(d') = d'$
  4. Induction: $\forall i \in \mathbb{N} : f^i(\bot) \sqsubseteq f^i(d') = d'$
  5. $\sup\{f^i(\bot) : i \in \mathbb{N}\} \sqsubseteq d'$

Induction over $\mathbb{N}$

Induction's principle:

$$(\forall X \subseteq \mathbb{N} : ((0 \in X \wedge (\forall x \in X : x \in X \rightarrow x + 1 \in X))) \rightarrow X = \mathbb{N})$$

Correctness:

1. Let's assume no, so $\exists X \subseteq \mathbb{N} : \mathbb{N} \setminus X \neq \emptyset$
2. Let y be minimum in $\mathbb{N} \setminus X$ (with respect to $<$).
3. $y \neq 0$
4. $y - 1 \in X \wedge y \notin X$
5. Contradiction

Induction over $\mathbb{N}$ (Alternative)

Induction's principle:

$$(\forall X \subseteq \mathbb{N} : (\forall x \in \mathbb{N} : \mathrm{sec}(x) \subseteq X \rightarrow x \in X) \rightarrow X = \mathbb{N})$$

Correctness:

1. Let's assume no, so $\exists X \subseteq \mathbb{N} : \mathbb{N} \setminus X \neq \emptyset$
2. Let y be minimum in $\mathbb{N} \setminus X$ (with respect to $<$).
3. $\mathrm{sec}(y) \subseteq X, y \notin X$
4. Contradiction

Well-founded induction

Induction's principle: Let $(Z, \leq)$ be a well-founded partial order.

$$(\forall X \subseteq Z : (\forall x \in Z : \mathrm{sec}(x) \subseteq X \rightarrow x \in X) \rightarrow X = Z)$$

Correctness:

1. Let's assume no, so $Z \setminus X \neq \emptyset$
2. Let z be a minimum in $Z \setminus X$ (in respect of $\leq$).
3. $\mathrm{sec}(z) \subseteq X, z \notin X$
4. Contradiction

FP-Induction: Proving properties of fixpoints

Induction's principle: Let $(D, \sqsubseteq)$ CPO, $f : D \to D$ continuous.

$$(\forall X \subseteq D \text{ admissible} : (\bot \in X \wedge (\forall y : y \in X \to f(y) \in X)) \to \mu f \in X)$$

Correctness: Let $X \subseteq D$ admissible.

$$\begin{array}{rcll}
\mu f \in X & \Leftrightarrow & \sup\{f^i(\bot) : i \in \mathbb{N}\} \in X & \text{(FP-theorem)} \\
& \Leftarrow & \forall i \in \mathbb{N} : f^i(\bot) \in X & (X \text{ admissible }) \\
& \Leftarrow & \bot \in X \wedge (\forall n \in \mathbb{N} : f^n(\bot) \in X \to f(f^n(\bot)) \in X) & \\
& & & \text{(Induction } \mathbb{N}) \\
& \Leftarrow & \bot \in X \wedge (\forall y \in X \to f(y) \in X) & \text{(Ass.)}
\end{array}$$

Problem

Exercise 4.3. *Let* $(D, \sqsubseteq)$ *CPO with*

- $X = Y = \mathbb{N}$
- $D = X \nrightarrow Y$: *set all partial functions* f *with* $\text{dom}(f) \subseteq X$ *and* $\text{cod}(f) \subseteq Y$.
- *Let* $f, g \in X \nrightarrow Y$.

$$f \sqsubseteq g \text{ iff } \text{dom}(f) \subseteq \text{dom}(g) \wedge (\forall x \in \text{dom}(f) : f(x) = g(x))$$

Consider

$$\begin{array}{rrcl} F: & D & \rightarrow & \mathcal{P}(\mathbb{N} \times \mathbb{N}) \\ & g & \mapsto & \begin{cases} \{(0,1)\} & g = \emptyset \\ \{(x, x \cdot g(x-1)) : x - 1 \in \text{dom}(g)\} \cup \{(0,1)\} & \text{otherwise} \end{cases} \end{array}$$

Problem

Prove:

1. $\forall g \in D : F(g) \in D$, i.e. $F : D \rightarrow D$
2. $F : D \rightarrow D$ continuous
3. $\forall n \in \mathbb{N} : \mu F(n) = n!$

Note:

- μF can be understood as the semantics of a function's definition

$$\begin{array}{l}\text{function}\,\text{Fac}(n : \mathbb{N}_\perp) : \mathbb{N}_\perp =_{\text{def}} \\ \quad \text{if } n = 0 \text{ then } 1 \\ \quad \text{else } n \cdot \text{Fac}(n-1)\end{array}$$

- Keyword: 'derived functions' in ASM

Problem

Exercise 4.4. *Prove:* *Let* $G = (V, E)$ *be an infinite directed graph with*

- *G has finitely many roots (nodes without incoming edges).*
- *Each node has finite out-degree.*
- *Each node is reachable from a root.*

There exists an infinite path that begins on a root.

Distributed ASM

Definition 4.5. *A DASM A over a signature (vocabulary) Σ is given through:*

- *A distributed programm Π_A over Σ.*
- *A non-empty set I_A of initial states*
  An initial state defines a possible interpretation of Σ over a potential infinite base set X.

A contains in the signature a dynamic relation's symbol $AGENT$, that is interpreted as a finite set of autonomous operating agents.

- *The behaviour of an agent a in state S of A is defined through $program_S(a)$.*
- *An agent can be ended through the definition of $program_S(a) := undef$ (representation of an invalid programm).*

Partially ordered runs

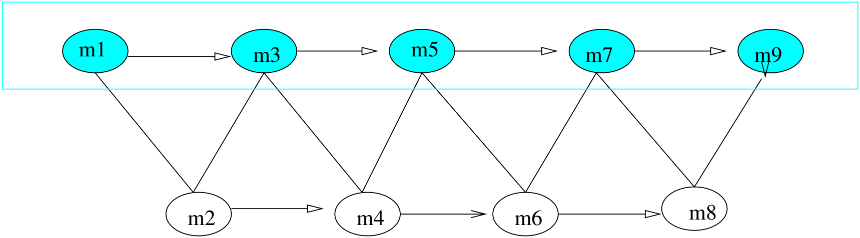
A run of a distributed ASM A is given through a triple $\varrho \rightleftharpoons (M, \lambda, \sigma)$ with the following properties:

1. M is a partial ordered set of "moves", in which each move has only a finite number of predecessors.
2. λ is a function on M, that assigns an agent to each move, so that the moves of a particular agent are always linearly ordered.
3. σ asociates a state of A with each finite initial segment Y of M. Intended meaning:: $\sigma(Y)$ is the "result of the execution of all moves in Y". $\sigma(Y)$ is an initial state when Y is empty.
4. The coherence condition is satisfied:
   If max is a set of maximal elements in a finite initial segment X of M and $Y = X \setminus max$, then for $x \in max$:: $\lambda(x)$ is an agent in $\sigma(Y)$ and we get $\sigma(X)$ from $\sigma(Y)$ by firing $\{\lambda(x) : x \in max\}$ (their programs) in $\sigma(Y)$.

Comment, example

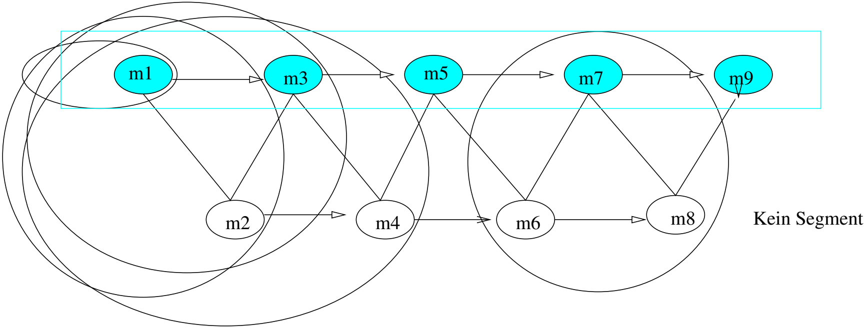
The agents of A modell the concurrent control-threads in the execution of Π_A.
A run can be seen as the common part of the history of the same computation from the point of view of multiple observers.

The role of λ:



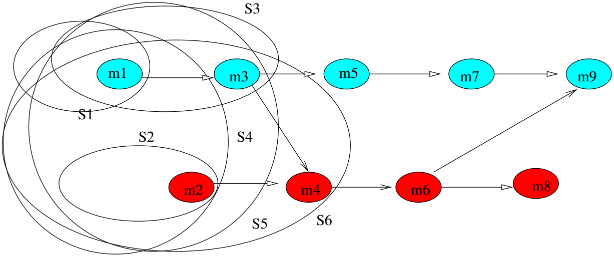
Comment, example (cont.)

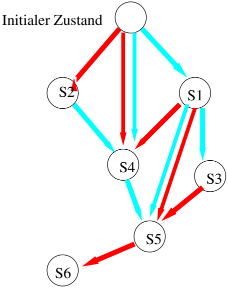
The role of σ: Snap-shots of the computation are the initial segments of the partial ordered set M. To each initial segment a state of A is assigned (interpretation of Σ), that reflects the execution of the programs of the agents that appear in the segment.
$\rightsquigarrow$"Result of the execution of all the moves" in the segment.



Coherence condition, example

If *max* is a set of maximal elements in a finite initial segment X of M and $Y = X \setminus max$, then for $x \in max$:: $\lambda(x)$ is an agent in $\sigma(Y)$ and we get $\sigma(X)$ from $\sigma(Y)$ by firing $\{\lambda(x) : x \in max\}$ (their programs) in $\sigma(Y)$.





Consequences of the coherence condition

Lemma 4.6. *All the linearizations of an initial segment (i.e. respecting the partial ordering) of a run ϱ lead to the same "final" state.*

Lemma 4.7. *A property P is valid in all the reachable states of a run ϱ, iff it is valid in each of the reachable states of the linearizations of ϱ.*

Simple example

Example 4.8. *Let $\{door, window\}$ be propositional-logic constants in the signature with natural meaning:*
$door = true$ means " door open " and analog for window.

The program has two agents, a door-manager d and a window-manager w with the following programs:

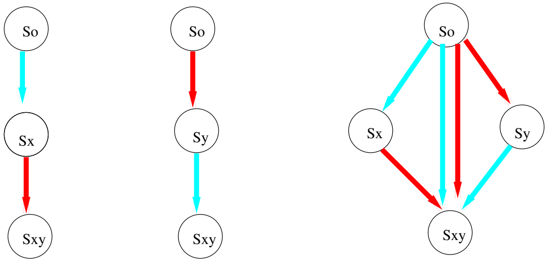
$program_d = door := true$ // *move x*
$program_w = window := true$ // *move y*

In the initial state S_0 let the door and window be closed, let d and w be in the agent set.

Which are the possible runs?

Simple example (Cont.)

Let $\varrho_1 = ((\{x, y\}, x < y), id, \sigma)$, $\varrho_2 = ((\{x, y\}, y < x), id, \sigma)$,
$\varrho_3 = ((\{x, y\}, <>), id, \sigma)$ (coarsest partial order)

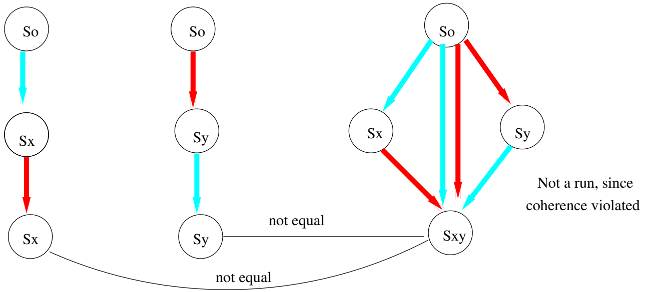


Variants of simple example

The program consists of two agents, a door-Manager d and a window-manager w with the following programs:

$program_d = if \neg window\ then\ \ door := true$ // move x
$program_w = if \neg door\ then\ \ window := true$ // move y

In the initial state S_0 let the door and window be closed, let d and w be in the agent set. How do the runs look like? Same ϱ's as before.



More variations

Exercise 4.9. *Consider the following pair of agents* $x, y \in \mathbb{N}$ $\quad$ *($x = 2, y = 1$ in the initial state)*

1. $a = x := x + 1$ *and* $b = x := x + 1$
2. $a = x := x + 1$ *and* $b = x := x - 1$
3. $a = x := y$ *and* $b = y := x$

Which runs are possible with partial-ordered sets containing two elements?

Try to characterize all the runs.

More variations

Consider the following agents with the conventional interpretation:

1. $Program_d =$ $if\ \neg window\ \ then\ \ door := true$ //move x
2. $Program_w =$ $if\ \neg door\ \ then\ \ window := true$ //move y
3. $Program_l =$ $if\ \neg light \wedge (\neg door \vee \neg window)\ \ then$ //move z
   $light := true$
   $door := false$
   $window := false$

Which end states are possible, when in the initial state the three constants are false?

Further exercises

Consumer-producer problem: Assume a single producer agent and two or more consumer agents operating concurrently on a global shared structure. This data structure is linearly organized and the producer adds items at the one end side while the consumers can remove items at the opposite end of the data structure. For manipulating the data structure, assume operations *insert* and *remove* as introduced below.

$$insert : \quad Item \times \quad ItemList \rightarrow \quad ItemList$$
$$remove : \quad ItemList \rightarrow \quad (Item \times \quad ItemList)$$

(1) Which kind of potential conflicts do you see?
(2) How does the semantic model of partially ordered runs resolve such conflicts?

Environment

Reactive systems are characterized by their interaction with the environment. This can be modeled with the help of an environment-agent. The runs can then contain this agent (with λ), λ must define in this case the update-set of the environment in the corresponding move.
The coherence condition must also be valid for such runs.

For externally controlled functions this surely doesn't lead to inconsistencies in the update-set, the behaviour of the internal agents can of course be influenced. Inconsistent update-sets can arise in shared functions when there's a simultaneous execution of moves by an internal agent and the environment agent.

Often certain assumptions or restrictions (suppositions) concerning the environment are done.
In this aspect there are a lot of possibilities: the environment will be only observed or the environment meets stipulated integrity conditions.

Time

The description of real-time behaviour must consider explicitly time aspects. This can be done successfully with help of timers (see SDL), global system time or local system time.

- The reactions can be instantaneous (the firing of the rules by the agents don't need time)
- Actions need time

Concerning the global time consideration, we assume, that there is on hand a linear ordered domain *TIME*, for instance with the following declarations:

$$\mathit{domain} \quad (\mathit{TIME}, \leq), \quad (\mathit{TIME}, \leq) \subset (\mathbb{R}, \leq)$$

In these cases the time will be measured with a discrete system watch: e.g.

$$\mathit{monitored} \quad \mathit{now} :\rightarrow \mathit{TIME}$$

ATM (Automatic Teller Machine)

Exercise 4.10. *Abstract modeling of a cash terminal:*
Three agents are in the model: ct-manager, authentication-manager, account-manager. To withdraw an amount from an account, the following logical operations must be executed:

1. *Input the card (number) and the PIN.*
2. *Check the validity of the card and the PIN (AU-manager).*
3. *Input the amount.*
4. *Check if the amount can be withdrawn from the account (ACC-manager).*
5. *If OK, update the account's stand and give out the amount.*
6. *If it is not OK, show the corresponding message.*

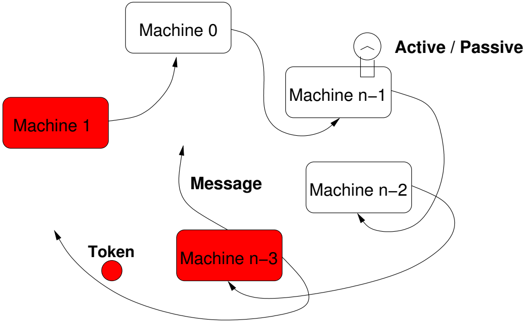
Implement an asynchronous communication model in which timeouts can cancel transactions .

Distributed Termination Detection

Example 4.11. *Implement the following termination detection protocol:*

A passive machine becomes active, iff it receives a message from another machine.

Only active machines can send messages.



Edsger W. Dijkstra, W. H. J. Feijen, and A.J.M. van Gasteren. Derivation of a Termination Detection Algorithm for Distributed Computations. IPL 16 (1983).

Assumptions for distributed termination detection

Rules for a probe

Rule 0 When active, $Machine_{i+1}$ keeps the token; when passive, it hands over the token to $Machine_i$.

Rule 1 A machine sending a message makes itself red.

Rule 2 When $Machine_{i+1}$ propagates the probe, it hands over a red token to $Machine_i$ when it is red itself, whereas while being white it leaves the color of the token unchanged.

Rule 3 After the completion of an unsuccessful probe, $Machine_0$ initiates a next probe.

Rule 4 $Machine_0$ initiates a probe by making itself white and sending to $Machine_{n-1}$ a white token.

Rule 5 Upon transmission of the token to $Machine_i$, $Machine_{i+1}$becomes white. (Notice that the original color of $Machine_{i+1}$ may have affected the color of the token).

Distributed Termination Detection: Procedure

Signature:

static

$COLOR = \{red, white\} \quad TOKEN = \{redToken, whiteToken\}$
$MACHINE = \{0, 1, 2, \ldots, n-1\}$
$next : MACHINE \rightarrow MACHINE$
e.g. with $next(0) = n-1, next(n-1) = n-2, \ldots, next(1) = 0$

controlled

$color : MACHINE \rightarrow COLOR \quad token : MACHINE \rightarrow TOKEN$
$RedTokenEvent, WhiteTokenEvent : MACHINE \rightarrow BOOL$

monitored $Active : MACHINE \rightarrow BOOL$
$SendMessageEvent : MACHINE \rightarrow BOOL$

Distributed Termination Detection: Procedure

Macros: (Rule definitions)

- $ReactOnEvents(m : MACHINE) =$
  - if $RedTokenEvent(m)$ then
    - $token(m) := redToken$
    - $RedTokenEvent(m) := undef$
  - if $WhiteTokenEvent(m)$ then
    - $token(m) := whiteToken$
    - $WhiteTokenEvent(m) := undef$
  - if $SendMessageEvent(m)$ then $color(m) := red$ Rule 1
- $Forward(m : MACHINE, t : TOKEN) =$
  - if $t = whiteToken$ then
    - $WhiteTokenEvent(next(m)) := true$
  - else
    - $RedTokenEvent(next(m)) := true$

Distributed Termination Detection: Procedure

Programs

- $RegularMachineProgram =$

  $ReactOnEvents(me)$
  $if \neg\ Active(me) \wedge\ token(me) \neq\ undef\ \ then$ Rule 0
  $\quad InitializeMachine(me)$ Rule 5
  $\quad if\ \ color(me) =\ red\ \ then$
  $\quad\quad Forward(me, redToken)$ Rule 2
  $\quad else$
  $\quad\quad Forward(me, token(me))$ Rule 2

- With $InitializeMachine(m : MACHINE) =$

  $token(m) := undef$
  $color(m) := white$

Distributed Termination Detection: Procedure

Programs

- $SupervisorMachineProgram =$

$$
\begin{array}{l}
ReactOnEvents(me) \\
if \neg\ Active(me) \wedge\ token(me) \neq\ undef\quad then \\
\quad if\quad color(me) =\ white \wedge\ token(me) =\ whiteToken\quad then \\
\quad\quad ReportGlobalTermination \\
\quad else\quad \text{Rule 3} \\
\quad\quad InitializeMachine(me)\quad \text{Rule 4} \\
\quad\quad Forward(me, whiteToken)\quad \text{Rule 4}
\end{array}
$$

Distributed Termination Detection

Initial states

$$\exists m_0 \in MACHINE$$
$$(program(m_0) = SupervisorMachineProgram \wedge$$
$$token(m_0) = redToken \wedge$$
$$(\forall m \in MACHINE)(m \neq m_0 \Rightarrow$$
$$(program(m) = RegularMachineProgram \wedge token(m) = undef)))$$

Environment constraints For all the executions and all linearizations holds:

$$\mathbf{G}\ (\forall m \in MACHINE)$$
$$(SendMessageEvent(m) = true \Rightarrow (\mathbf{P}(Active(m)) \ \wedge Active(m)))$$
$$\wedge \ ((Active(m) = true \wedge \mathbf{P}(\neg Active(m)) \Rightarrow$$
$$(\exists m' \in MACHINE) \ (m' \neq m \wedge \ SendMessageEvent(m'))))$$

Nextconstraints

Distributed Termination Detection

Correctness of the abstract version: Dijkstra

Suppositions: The machines constitute a closed system, i.e. messages can only be dispatched among each other (no outside messages). The system in the initial state can have any color and several machines can be active. The token is located in the 0'th. machine. The given rules describe the transfer of the token and the coloration of the machines upon certain activities.

The task is to determine a state in which all the machines are passive (not active). This is a stable state of the system, because only active machines can dispatch messages and passive machines can only become active by receiving a message.

The invariant: Let t be the position on which the token is, then following invariant holds

$$(\forall i : t < i < n \quad Machine_i \text{ is passive}) \vee (\exists j : 0 \leq j \leq t \quad Machine_j \text{ is red}) \vee (Token \text{ is red})$$

Distributed Termination Detection

$(\forall i : t < i < n \;\; Machine_i$ is passive$) \vee (\exists j : 0 \leq j \leq t \;\; Machine_j$ is red$) \vee$ $(Token$ is red$)$

Correctness argument
When the token reaches $Machine_o$, $t = 0$ and the invariant holds.
If
$(Machine_o$ is passive$) \wedge (Machine_o$ is white$) \wedge (Token$ is white$)$
then
$(\forall i : 0 < i < n \;\; Machine_i$ is passive$)$ must hold, i.e. termination.

Proof of the invariant Induction over t:
The case t = n - 1 is easy.
Assume the invariant is valid for $0 < t < n$, prove it is valid for $t - 1$.

Distributed Termination Detection

Is the invariant valid in all the states of all the linearizations of the runs of the DASM ? **No**

- **Problem 1** The red coloration of an active machine (that forwards a message) occurs in a later state. It should occur in the same state in which the message-receiving machine turns active. (Instantaneous message passing)
  Solution *color* is a shared function. Instead of using *SendMessageEvent*(m) to set the color, it will be set by the environment: $color(m) = red$.
- **Problem 2** There are states in which none of the machines has the token:: The machine that has the token, initializes itself and sets an event, that leads to a state in which none of the machines has the token.
  Solution Instead of using *FarbTokenEvent* to reset, it is directly properly set: $token(next(m))$.
- **Result** More abstract machine. The environment controls the activity of the machines, message passing and coloration.

Refinement's concepts for ASM's

Question: Is in the termination detection example the given DASM a refinement of the abstracter DASM? $\rightsquigarrow$

General refinement concepts for ASM's

- Refinements are normally defined for BASM, i.e. the executions are linear ordered runs, this makes the definition of refinements easier.
- Refinements allow abstractions, realization of data and procedures.
- ASM refinements are usually problem-oriented: Depending on the application a flexible notion of refinement should be used.
- Proof tasks become structured and easier with help of correct and complete refinements.

See ASM-Buch.
Example Shortest Path