PhD Summer School
Formal Methods for System Analysis in Informatics
Druskininkai, LT, May 2007

Formal specification, observability, discreteness, and languages

Bernd Baumgarten

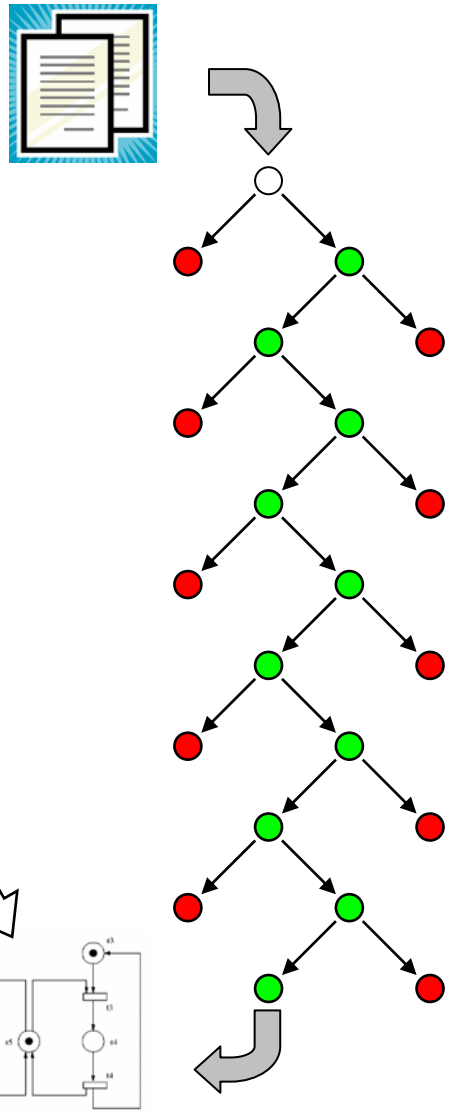
Fraunhofer SIT
Rheinstr. 75, 64295 Darmstadt, Germany
bernd.baumgarten@sit.fraunhofer.de
http://private.sit.fhg.de/~baumgart/
+49 6151 869263

The purpose of a formal specification

- precise **blueprint** for system implementations,
  part of a contract, directives, software engineering
- theoretical model to check the fulfillment of **requirements** by mathematical **analysis**
  behaviour of abstract mechanisms
- input for automatic **prototype** generation
  rapid prototyping
- input for automatic **test case** generation
  used to test the conformance of implementations, to automate a bulk job
- simply **to get your ideas straight**
  our "recovery line experience" – 99.9% error expectancy!

The “1984 recovery line spec experience”

- 1980’s topic: **Recovery line algorithm (RLA),** a **distributed** algorithm, or protocol.
- **simple** natural language **specification**, about ½ page.
- our aim: **formalize** it and **prove** it correct
- (new on top: **specification** of the **aim** of RLA)
- during formalization: ±10 binary choices
- Each time: analysis of both alternatives
- Each time, one alternative was bad **!**
- Chance of success of “**straightforward** programming”: $\mathbf{2^{-10}} < 0.1\%$



Using a formal specification

… is like thorough testing:

Everybody agrees it is **important**,

but **nobody** wants to spend **money** on it.

Yes, security/safety is expensive, but …

it may simply be **essential**.

Think of …

medical applications

air traffic control

power plants

etc

What do specifications describe?

A **specification** describes ...

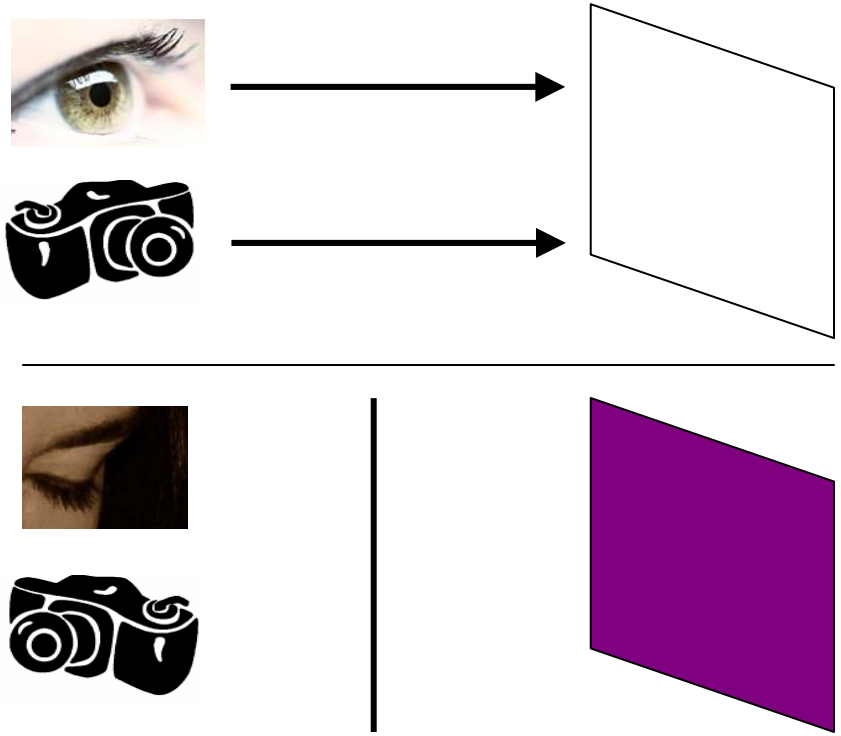
permitted, required, forbidden } system { observations, behaviour(s), properties.

Practical + mathematical + philosophical analysis …

- What is the inside color of the casing of your PC? *properties?*
- The story of the → paint supposed to switch colours *behaviour?*
- Forbidden is the complement of permitted *forbidden?*
- Requirements without timing: did you ever lend money? or use a vending machine? *required?*

→ It is sufficient to specify the **set of permitted**/desirable observations, or, more precisiely **observation protocols**.

Unobservable behaviour: the tale of the strange white paint



A crazy billionaire offered $1,000,000 to anybody who painted his sleeping room with a very special "**WP**" paint. It should look **white** as long as observed, but should turn **purple** as soon as nobody, or no instrument, watched it.

A painter bought plain white paint for $50, painted the room, obtained the million, and the billionaire was happy. But, consciuous-stricken, the painter donated all the money to charity. The next morning, he read in the newspaper that the white paint factory had taken part in secret experiments for the military (camouflage etc.). They had for months sold WP paint as plain white paint, to test if anybody would notice …

Painter: ☹ By the way: What if the colour sold had turned green instead of purple?

No requirements without timing

With timing → violation **observable**!

Example

You pay something in advance. Your contractor promises to deliver.

Then do until dead:

(You ask after 4 weeks.
Contractor says: not yet, but soon.
You cannot even complain, as no deadline was specified.)

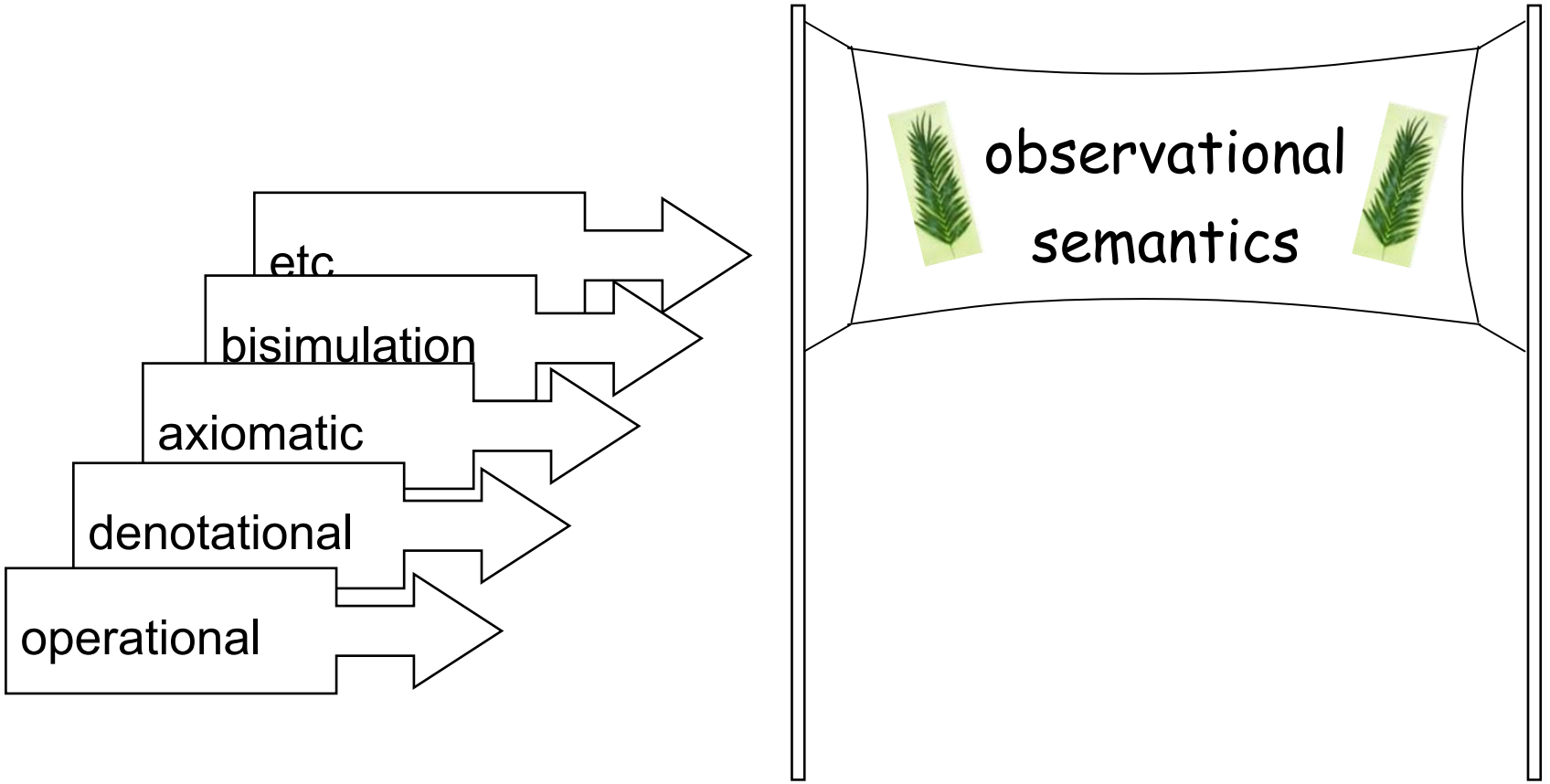
Way out

- Specify a deadline, too.
- Then, if things run bad, you can **observe** (prove) that your specification was violated, and claim your money back.

A very common omission!

German civil case law →
you may specify a reasonable deadline later, if you forgot to do so initially.

Approaches to Semantics



Empirism

Empirism
(Pragmatism, Positivism, Materialism, Behaviourism, Realism, ...)

→

A **system** „is"
whatever about it
you **are able to observe,**
or choose to observe
– directly oder indirectly.

An application of empirism

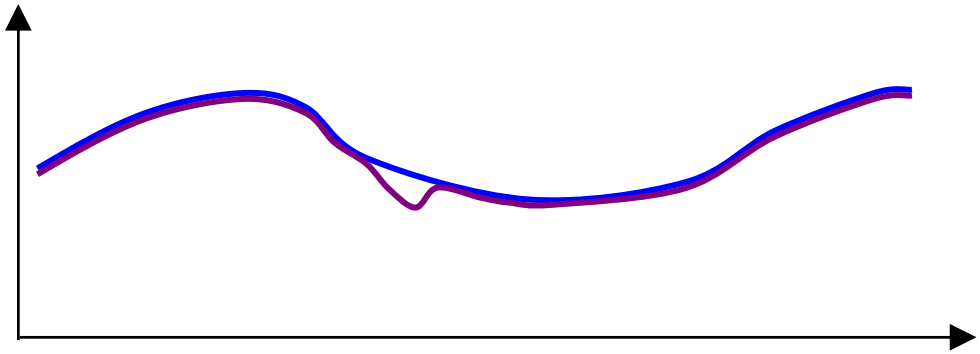
Philosophical & physical debate:

Do truly discrete systems exist?
Do truly continuous systems exist?

→

We may consider
systems
(discrete oder continuous!)
– by our own decision
& capability of abstraction –
als **discretely**
observable and controllable.

Discrete observations of continuous systems



Specify a real function $y = \mathbf{f}(x)$.

Buy an alleged implementation $y = \mathbf{g}(x)$.

Observe finitely many $x_1, x_2, \ldots, x_n$.

- What **can** you find out (if lucky)? : – $\mathbf{f} \neq \mathbf{g}$, $\mathrm{Max}(\mathbf{g}) > \mathrm{Max}(\mathbf{f})$, …
- What **can** you **not** find out? : – $\mathbf{f} = \mathbf{g}$, $\mathrm{Max}(\mathbf{g}) < \mathrm{Max}(\mathbf{f})$, $\int\mathbf{f} = \int\mathbf{g}$, $\int\mathbf{f} \neq \int\mathbf{g}$, …

Discrete *(ly observed&controlled)* systems

DORTS = **D**iscretely **O**bserved **R**eal **T**ime **S**ystem *sys*

natural DORTS semantics:

set *Obs*(*sys*) of possible **observations** (observation and control protocols *obs*)

$$obs = \big((\alpha, t_0), (v_1, t_1), (v_2, t_2), \ldots, (v_n, t_n), (\omega, t_{n+1}) \big)$$

? **finite** sequence

$n = 0, 1, \ldots$

$v_k \in Events$ **relevant visible atomic events** (data objects) – initiated by observer, system or both

?

$t_k \in I\!R, \quad t_k \le t_{k+1}$ **Moments** of atomic events in abs. time

α, ω ? **Begin** resp. **end** of observation (**optional**)

Specification, implementation, and conformance

With our **simple view** of systems and specifications:

implementation = sys_1
specification = sys_2

conformance: $Obs(sys_1) \subseteq Obs(sys_2)$

Appropriate semantics →

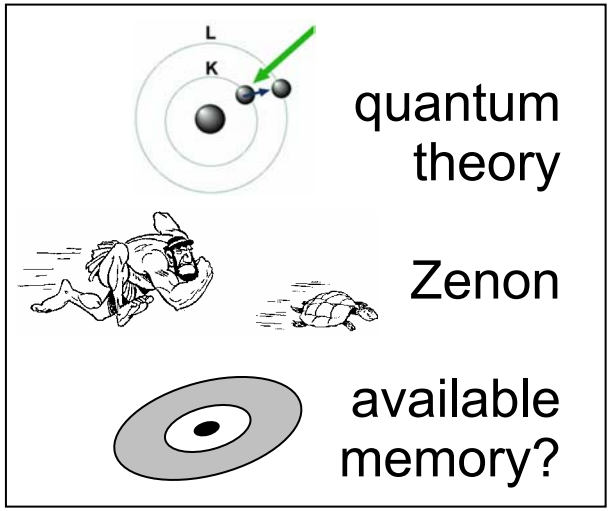
Conformance is as easy as  !

DORTS – questions

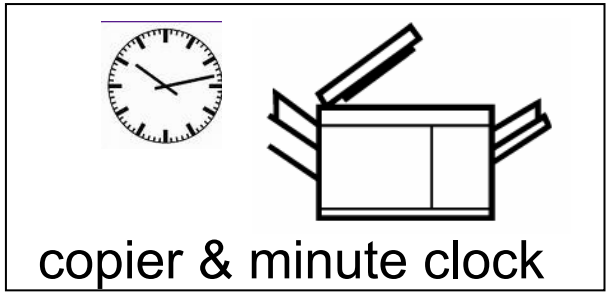
- Why only **finite** observations (observation protocol)?

Infinite duration? ➔ **Observer** 19XY – 20YX

Infinitely many meaningful events in finite time?



- Why only $t_k \leq t_{k+1}$ and not $t_k < t_{k+1}$ – can there be a meaningful order "at t_k"?



DORTS – simplifications sometimes taken

#1: $t_k \in \mathbb{N}$ (small time unit)

#2: $t_k < a,\ a \in \mathbb{R} \rightarrow t_k \leq b,\ b \in \mathbb{N}$
Replace bounds by natural numbers.

#3: $\alpha, \omega \in Events \rightarrow obs = ((v_1, t_1), (v_2, t_2), \ldots, (v_n, t_n))$

make them explicit
in the model !

Languages

Our topic: **discrete** systems

➔ **observation (protocol)** = a **finite sequence** of observation elements

observation element a data record, typically stating …

Who – when – where – what?

or Who – from ... to ... – where – nothing?

A (formal) specification defines a **(formal) language**,

the set of permitted observation sequences

over the **alphabet** of observation elements

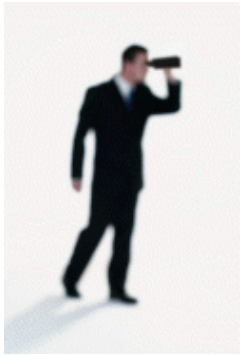
that are **imaginable and of interest**.

Untimed languages and models

- were often preferred as an object of research
- maybe because they helped to avoid some nasty technicalities.

Compare with:

- losing your key here in a dark place
- but looking for it there under the nearby lantern
- because it is easier to see something there …



Language notions (basic)

alphabet: $A = \{s_1, \ldots, s_n\}$

symbol: $s_i \in A$

word: $w \in A^* = \{a_1 \ldots a_k \mid k \in I\!N_0, \forall 1 \leq i \leq k : a_i \in A\}$

empty word: ε $(k = 0)$

#(a,w): the number of occurrences of *a* in *w*

concatenation: $u \circ v = uv$

prefix of *w*: a *u* with $w = uv$

***Pref*(*w*)**: the set of all prefixes of *w*

language (over *A*): a set of words $L \subseteq A^*$

More basic language notions

prefix lang. of L: $Pref(L) := \bigcup_{w \in L} Pref(w)$

prefix-closed: $Pref(L) \subseteq L$.

Kleene star and power notation

* means "zero, one or several times"

$L^* := \{w_1 \ldots w_n \mid n \geq 0 \wedge \forall 0 < i \leq n : w_i \in L\}$,

a^* with $a \in A$ means $\{a\}^*$,

w^* with $w \in A^*$ means $\{w\}^*$,

a^k means $a \ldots a$ (k times).

A – perhaps not so common – language notion

quotient of *L* over *u*:

$$u^{-1}(L) := \{v \mid \exists w \in L : w = uv\}$$

= the “set of possible continuations after *u* in *L*”

Language definitions

Languages can be defined by means of

- listing their words – only if finite and not too big,
- automata and machine models,
- regular expressions,
- grammars,
- Petri nets,
- etc.

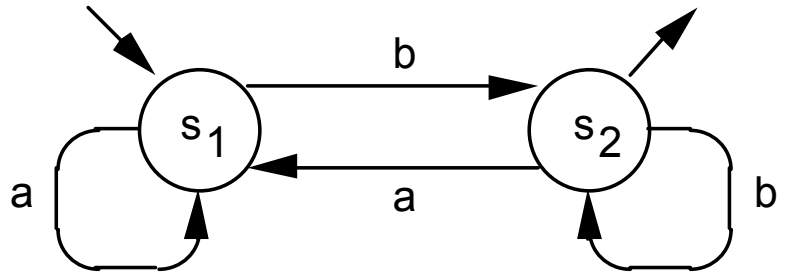
Automata (1)

A (**finite**) **acceptor** – or (**finite**) **deterministic automaton** – is a quintuple $Acc=(A, St, \delta, \alpha, \Omega)$ with

- an alphabet A;
- a (finite) set $St = \{z_1,\ldots,z_m\}$, the elements of which are called **states**;
- a **transition function** δ: $St \times A \rightarrow St$;
- an **initial state** $\alpha \in St$;
- a set $\Omega \subseteq St$ of **terminal states**.

Example

finite acceptor *Acc1*:
$A = \{a,b\}$; $St = \{s_1,s_2\}$;
$\alpha = s_1$; $\Omega = \{s_2\}$
$\delta = \{ ((s_1,a), s_1), ((s_1,b), s_2),$
$((s_2,a), s_1), ((s_2,b), s_2) \}$



Automata (2)

δ is recursively "extended" as Δ, to operate on words over A, by

$\Delta(s_i, \varepsilon) := s_i$ and

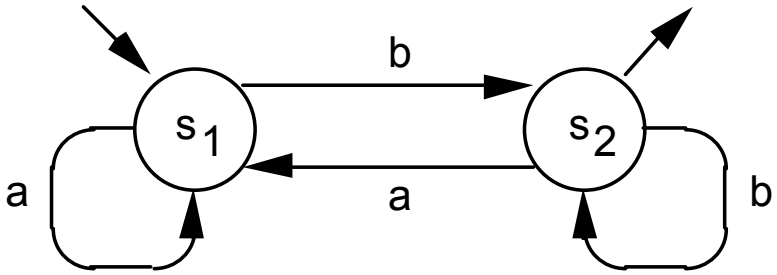
$\Delta(s_i, w) = s_j \wedge \delta(s_j, a) = s_k \Rightarrow \Delta(s_i, wa) := s_k$ for $w \in A^*$, $a \in A$

Instead of $\Delta(s_i, w) := s_j$ we sometimes write s_i—w–>s_j .

An acceptor *Acc* **accepts** (defines) the language

$L(Acc) := \{w \in A^* \mid \exists s \in \Omega: \alpha$—$w$–>$s\}$

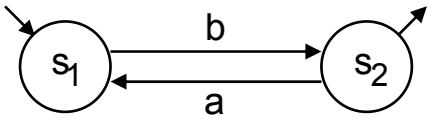
Exercise: Which words does *Acc1* accept?



Automata (3)

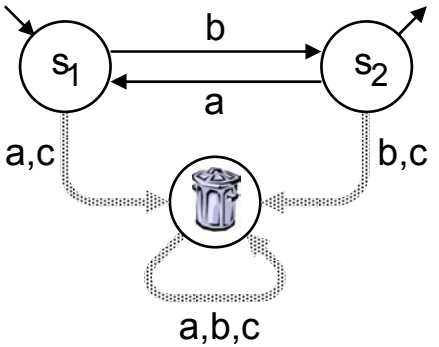
Incompletely written acceptors:

Acc2



over {*a,b,c*}

means *Acc3*



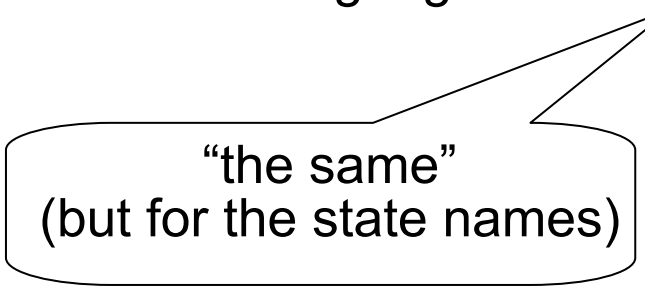
The canonical acceptor

Canonical acceptor *Acc*(*L*), accepts *L* (over *A*).

- alphabet *A*: (if open: set of all symbols occurring in at least one word);
- $St = \{u^{-1}(L) \mid u \in Pref(L)\} \cup \{\varnothing\}$ (state = possible "next behaviour");
- transition function $\delta : St \times A \to St$ with $\delta(s,a) := a^{-1}(s)$;
- the initial state $L = \varepsilon^{-1}(L)$;
- set of terminal states $\{s \in St \mid \varepsilon \in s\}$.

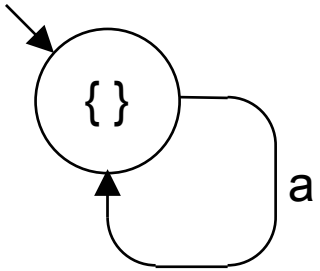
(1) *Acc*(*L*) is **minimal**:
no acceptor of *L*(over *A*) has fewer states than *Acc*(*L*).

(2) Minimal finite acceptors of the same language are **isomorphic**.



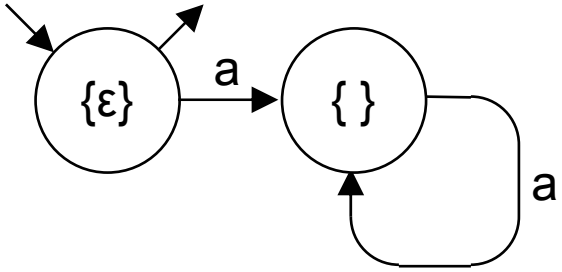
Canonical Acceptor Examples (over {a})

(a) *Acc*({ })

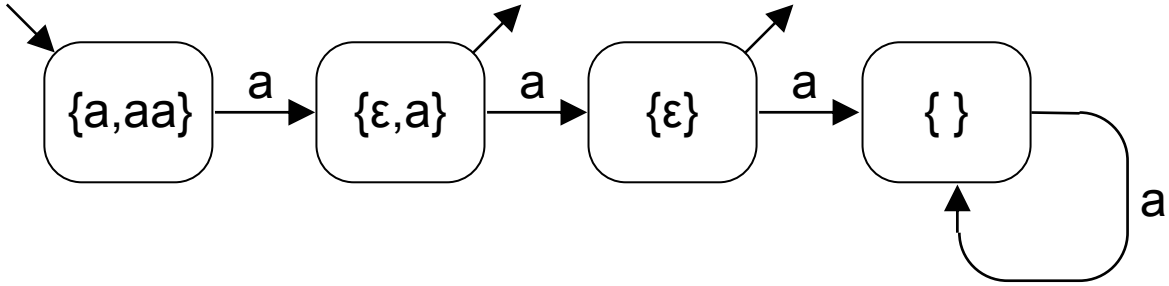


{ } ≈ 

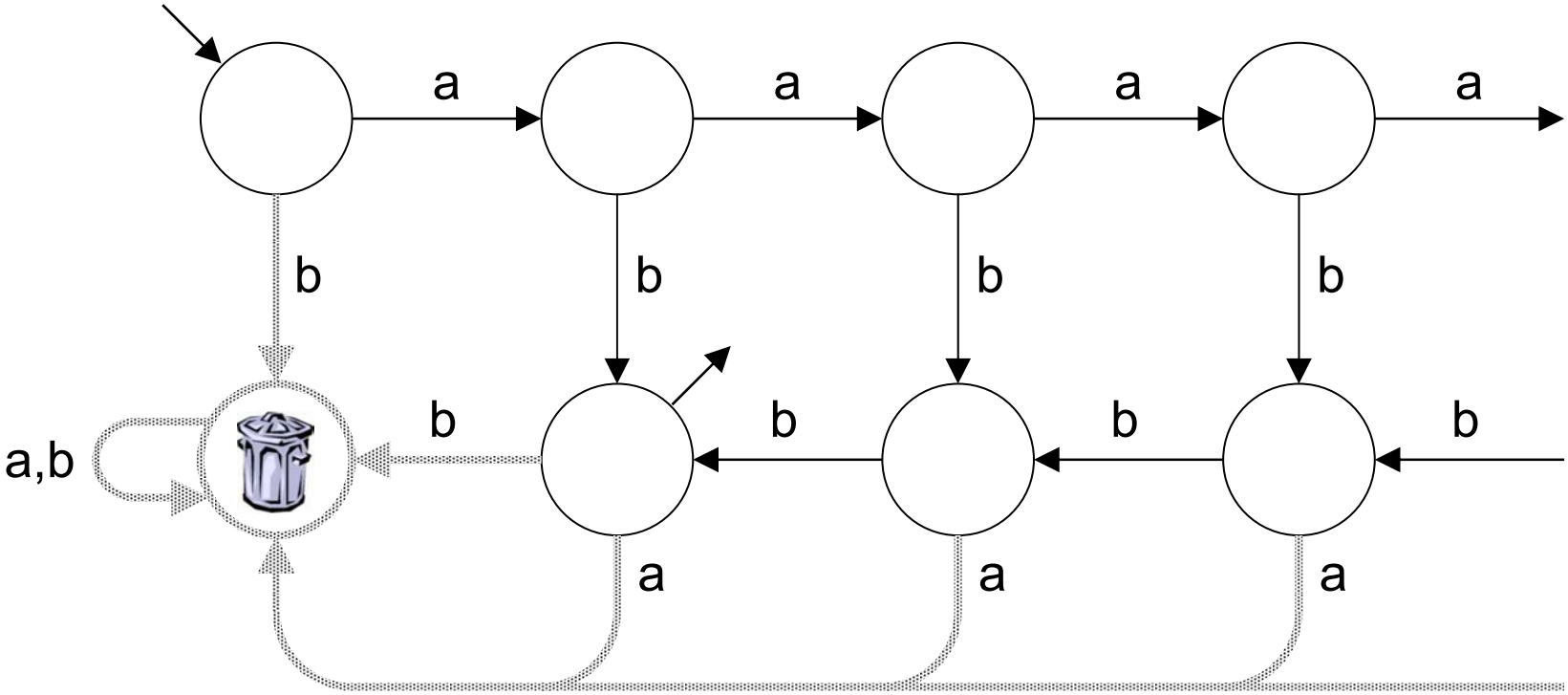
(b) *Acc*({ε})



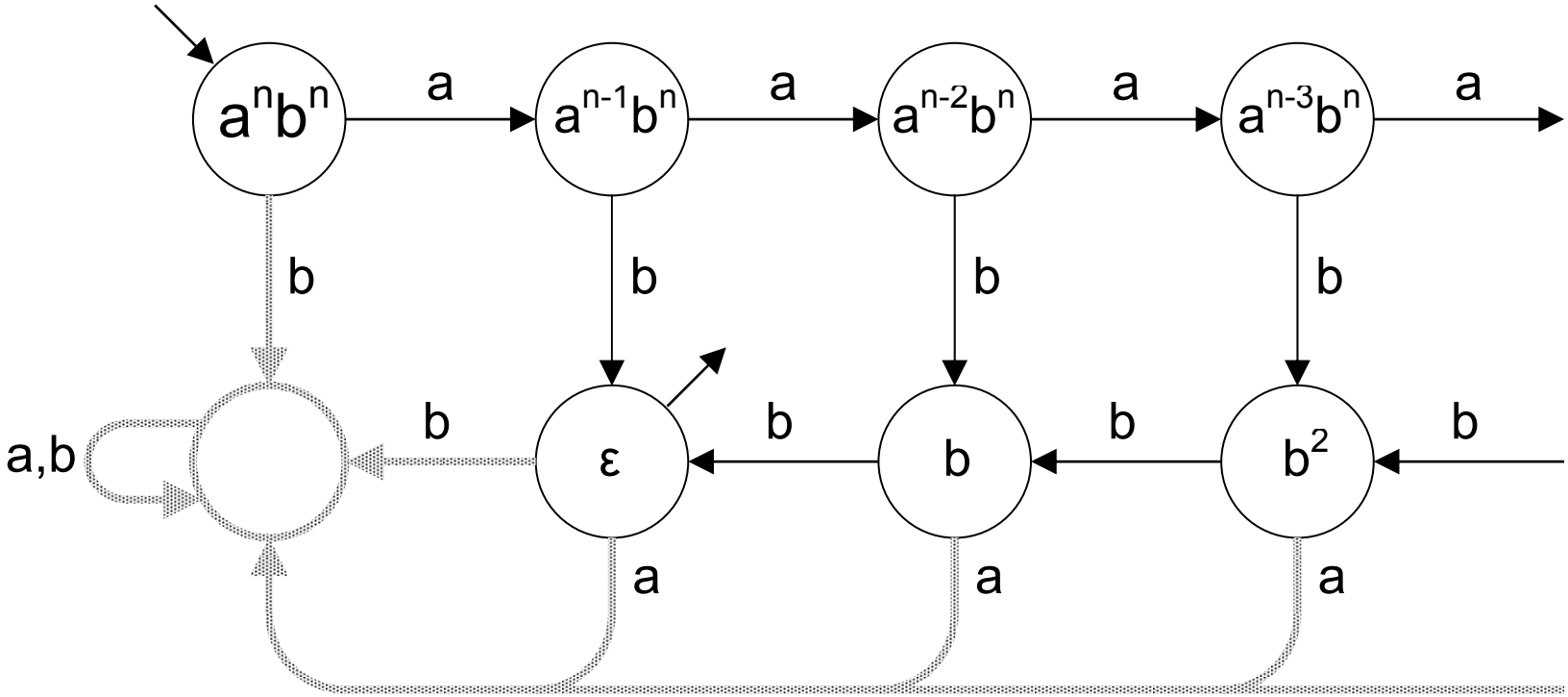
(c) *Acc*({a,aa})



An infinite canonical acceptor (over {a,b}) – for which language?



An infinite canonical acceptor for $\{a^n b^n \mid n=0,1,\ldots\}$



(not “regular”)

Grammars (1)

A **grammar** is a quadruple $G = (N, T, S, R)$ with

- an alphabet N of **nonterminal symbols**,
- an alphabet T of **terminal symbols**, $N \cap T = \varnothing$,
- a **start symbol** $S \in N$,
- a finite set $R \subset (N \cup T)^* \times (N \cup T)^*$ of **rules**.

⇐ often: restrictions

$(v,w) \in R$ is often written as $v \to w$;
and $(v,w),(v,w') \in R$ as $v \to w \mid w'$.

G defines inductively a language $\subseteq T^*$ of terminal symbol words,

$L(G) := H(G) \cap T^*$, where

the auxiliary language $H(G) \subseteq (N \cup T)^*$ is given by

- $S \in H(G)$
- $rvs \in H(G) \wedge v \to w \Rightarrow rws \in H(G)$.

Nice exercise: Write a grammar for {ww | w∈(a+b)*}.

Grammars (2)

G is called **right-linear** if all rules are of the form

$X \to wY$ or $X \to w$, with $X, Y \in N$ and $w \in T^*$.

Finite Acceptors and right-linear grammars define the same set of (**regular**) languages.

It is obvious how to obtain a right-linear grammar from a finite acceptor.

The other way around you may have to deal with non-deterministic state-machines.

Regular expressions

The set of **regular expressions** $Reg(A)$ over an alphabet A is defined inductively:

- $\emptyset, \varepsilon \in Reg(A)$
- $A \subseteq Reg(A)$ (symbol "=" word of length 1)
- $p, q \in Reg(A) \Rightarrow (p)^*, (p+q), (pq) \in Reg(A)$.

Every regular expression *reg* defines recursively a language $L(reg)$:

$L(\emptyset) := \emptyset$ $\quad L((p)^*) := (L(p))^*$

$L(\varepsilon) := \{\varepsilon\}$ $\quad L((p+q)) := L(p) \cup L(q)$

$L(a) := \{a\}$ for all $a \in A$ $\quad L((pq)) := \{vw \mid v \in L(p) \wedge w \in L(q)\}$

(Some parentheses may been omitted with binding priorities * > $\circ$ > +.)

Regular expressions also define the set of regular languages.

Expressivity

Let for a class C of formal models (e.g. grammars with terminal alph. {a,b,c})

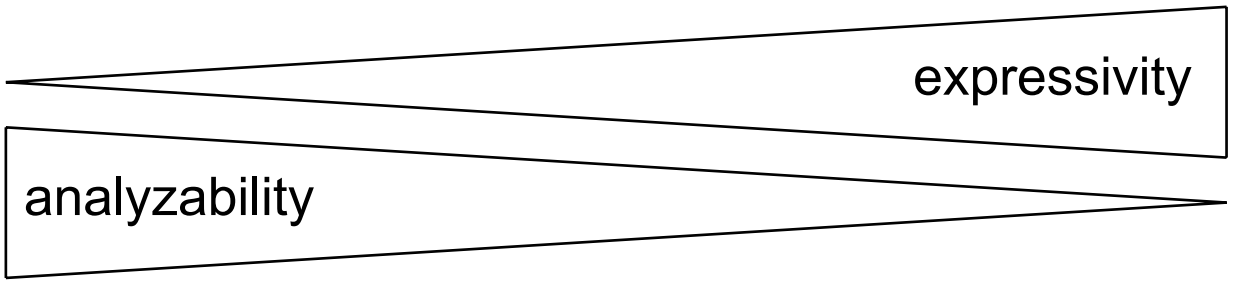
Lang(C) := the set of languages which are semantics of models in C

= the **expressivity** of C

C is (strictly) **more expressive** than D :⇔ Lang(C) ($\supset$) $\supseteq$ Lang(D).

Example: Lang(“abc-grammars”) $\supset$ Lang(“ab-acceptors”)

Greater expressivity is nice, but usually …



Literature

some complementary considerations about formal methods:

Jean François Monin: *Understanding Formal Methods*. Springer 2003

some useful books on formal languages:

Harry R.Lewis, Christos H. Papadimitriou: *Elements of the theory of computation (2nd ed.)*, Prentice-Hall, 1998

M. Davis: Computability, *Complexity and Languages: Fundamentals of Theoretical Computer Science (2nd ed.)*, Morgan Kaufmann, 1994

on discrete systems:

Christos H. Cassandras, St. Lafortune: *Introduction to Discrete Event Systems*, Kluwer, 1999

on observability:

→ cf. the works of Rudolf Carnap and by the “Wiener Kreis” in general

recovery line correctness proof with HLPN:

B. Baumgarten, P. Ochsenschlaeger: Modeling and Verification of a Checkpoint-Restart-Protocol, *Fehlertolerierende Rechensysteme*, Informatik-Fachberichte 84, Springer, 1984