Lezione 26 MSC
Espressività: Operatori aggiuntivi (2/2)

Roberto Gorrieri

Sequential composition

- Given two processes, *p* and *q*, we build a new process $p \cdot q$, whose behavior, intuitively, is composed of the behavior of *p* first, followed by that of *q*, if and when *p* terminates successfully.
- $p\downarrow$ is the predicate stating that p may successfully terminate

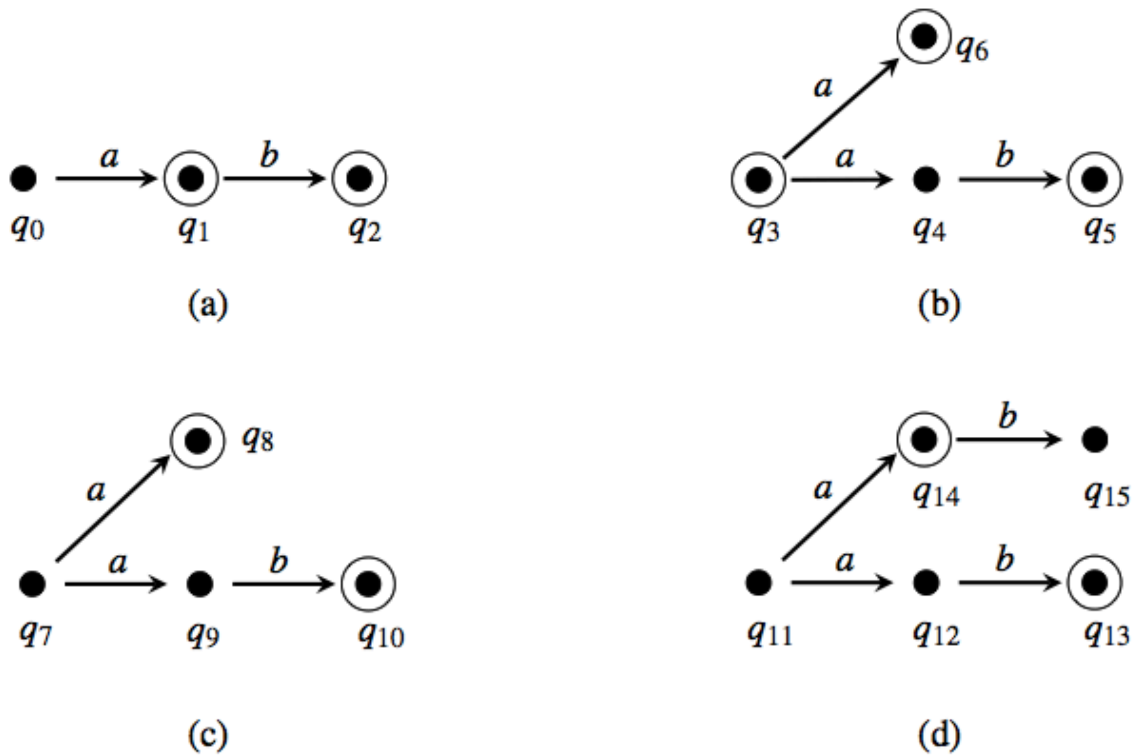
$$(\mathrm{Seq}_1)\ \frac{p \xrightarrow{\mu} p'}{p \cdot q \xrightarrow{\mu} p' \cdot q} \qquad (\mathrm{Seq}_2)\ \frac{p\downarrow \quad q \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} q'}$$

Sequential composition (2)

- This intuition can be formalized only if we allow for a more generous semantic model where the states are of two types: the *final* states, i.e., states that can immediately terminate successfully their execution, and the *non final* states, i.e., states that cannot immediately terminate successfully.
- Lts with final states (Q, A, $\rightarrow$, F)
- Additional parameter: set F of final states; a final state $p \in F$ can be equivalently represented as $p\downarrow$.

Lts with final states

- Additional parameter: set F of final states; a final state is graphically represented by a circle around the node.



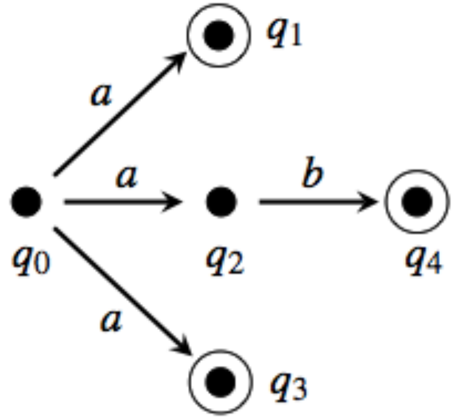
F-Behavioral equivalences (1)

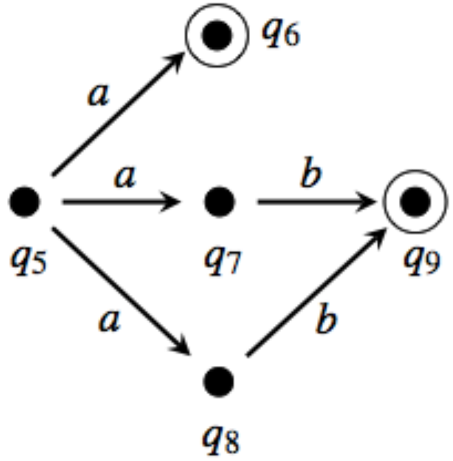
- An f-trace is a trace that ends in a final state.
- The lts's (a), (c) and (d) of the previous slide are f-trace equivalent.
- A completed f-trace is an f-trace ending in a deadlock.
- The lts's (a) and (d) are completed f-trace equivalent.
- A weak f-trace is a weak trace ending in a final state. A completed weak f-trace is a weak f-trace ending in a state that cannot perform any observable action.

F-Behavioral equivalences (2)

- An f-bisimulation R is a bisimulation that, additionally, for any pair (q, q') in R satisfies the type condition: q in F iff q' in F.

$$\sim_f = \bigcup \{ R \subseteq Q \times Q \mid R \text{ is a f-bisimulation} \}$$





F-Behavioral equivalences (3)

Definition 5.7. (Weak f-bisimulation) For any lts-f $TS = (Q, A \cup \{\tau\}, \rightarrow, F)$, a *weak f-bisimulation* is a relation $R \subseteq (Q \times Q)$ such that:

- R is a weak bisimulation over the underlying lts $(Q, A \cup \{\tau\}, \rightarrow)$;
- if $(q, q') \in R$ and $q \in F$, then $\exists q'' \in F$ such that $q' \stackrel{\varepsilon}{\Longrightarrow} q''$ and $(q, q'') \in R$;
- if $(q, q') \in R$ and $q' \in F$, then $\exists q'' \in F$ such that $q \stackrel{\varepsilon}{\Longrightarrow} q''$ and $(q'', q') \in R$.

State q is weakly f-bisimilar to q', denoted $q \approx_f q'$, if there exists a weak f-bisimulation R such that $(q, q') \in R$. $\square$

Definition 5.8. (Rooted weak f-bisimilarity) Given an lts-f $(Q, A \cup \{\tau\}, \rightarrow, F)$, two states q_1 and q_2 are rooted weak f-bisimilar, denoted $q_1 \approx_f^c q_2$, if for all $\mu \in A \cup \{\tau\}$

- $\forall q_1'$ such that $q_1 \xrightarrow{\mu} q_1'$, $\exists q_2'$ such that $q_2 \stackrel{\mu}{\Longrightarrow} q_2'$ and $q_1' \approx_f q_2'$
- $\forall q_2'$ such that $q_2 \xrightarrow{\mu} q_2'$, $\exists q_1'$ such that $q_1 \stackrel{\mu}{\Longrightarrow} q_1'$ and $q_1' \approx_f q_2'$
- $q_1 \in F$ iff $q_2 \in F$. $\square$

Finite BPA

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p+p \mid p \cdot p$$

$$(\text{Act}) \qquad \mu \xrightarrow{\mu} \mathbf{1}$$

$$(\text{Seq}_1)\ \frac{p \xrightarrow{\mu} p'}{p \cdot q \xrightarrow{\mu} p' \cdot q} \qquad (\text{Seq}_2)\ \frac{p\downarrow \quad q \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} q'}$$

$$(\text{Sum}_1)\ \frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'} \qquad (\text{Sum}_2)\ \frac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$$

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{p\downarrow}{(p+q)\downarrow} \qquad \frac{q\downarrow}{(p+q)\downarrow} \qquad \frac{p\downarrow \quad q\downarrow}{(p \cdot q)\downarrow}$$

Examples

- 1, 0+1, 1·1, $b\cdot c$+1, (a+1)·($b\cdot c$+1) are final states ↓
- While 0, a, 1·0, $a+b\cdot c$ are not final states

$$(a+\mathbf{1})\cdot b \xrightarrow{a} \mathbf{1}\cdot b \xrightarrow{b} \mathbf{1} \text{ and also } (a+\mathbf{1})\cdot b \xrightarrow{b} \mathbf{1}$$

Note that $Tr_f((a+\mathbf{1})\cdot b) = \{ab, b\}$, because only state $\mathbf{1}$ is final.

- EXERCISE: Compute the lts-f associated to the following finite BPA processes: $a\cdot(1+b)$, $a\cdot b+a+1$, $a+a\cdot b$ and $a\cdot b+a\cdot(b\cdot 0+1)$. Compare the resulting lts-f's with those in slide 4.

Algebraic properties

Choice operator:

$$
\begin{aligned}
p+(q+r) &\sim_f (p+q)+r \\
p+q &\sim_f q+p \\
p+\mathbf{0} &\sim_f p \\
p+p &\sim_f p
\end{aligned}
$$

Note that $\mathbf{0} \sim \mathbf{1}$, but $\mathbf{0} \not\sim_f \mathbf{1}$. Similarly, $p+\mathbf{1} \sim p$, but $p+\mathbf{1} \not\sim_f p$ in general. E.g., $a+\mathbf{1} \not\sim_f a$, because only $a+\mathbf{1}$ is final. □

Sequential composition: For f-bisimilarity

$$
\begin{array}{ll}
(i) & p\cdot(q\cdot r) \sim_f (p\cdot q)\cdot r \\
(ii) & \mathbf{0}\cdot p \sim_f \mathbf{0} \\
(iii) & \mathbf{1}\cdot p \sim_f p \\
(iv) & p\cdot \mathbf{1} \sim_f p \\
(v) & (p+q)\cdot r \sim_f p\cdot r + q\cdot r
\end{array}
$$

For f-trace equivalence

$$
\begin{array}{ll}
(i) & p\cdot \mathbf{0} =_{trf} \mathbf{0} \\
(ii) & r\cdot(p+q) =_{trf} r\cdot p + r\cdot q
\end{array}
$$

Congruence

- $\sim_f$ and $\approx_f$ are congruences for sequential composition
- $\approx_f$ is not a congruence for choice: tau $\cdot a \approx_f a$, but tau $\cdot a + b$ is not weakly f-bisimilar to $a+b$
- $\sim_f$ and $\approx_f^c$ are congruences for choice
- $\approx_f^c$ is the coarsest congruence contained in $\approx_f$

Theorem 5.4. *Assume that $fn(p) \cup fn(q) \neq \mathscr{L}$. Then $p \approx_f^c q$ if and only if, for all $r \in \mathscr{P}_{finBPA}$, $p + r \approx_f q + r$.*

BPA*: finite BPA + iteration

- BPA* is the language of regular expressions:

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p+p \mid p \cdot p \mid p^*$$

$$\frac{p \in \mathscr{P}_{BPA^*}}{p^* \downarrow} \qquad (\text{Star})\ \frac{p \xrightarrow{\mu} p'}{p^* \xrightarrow{\mu} p' \cdot p^*}$$

$$(a \cdot b)^* \xrightarrow{a} (\mathbf{1} \cdot b) \cdot (a \cdot b)^* \xrightarrow{b} \mathbf{1} \cdot (a \cdot b)^* \xrightarrow{a} (\mathbf{1} \cdot b) \cdot (a \cdot b)^*$$

$$a \cdot (b \cdot a)^* \xrightarrow{a} \mathbf{1} \cdot (b \cdot a)^* \xrightarrow{b} (\mathbf{1} \cdot a) \cdot (b \cdot a)^* \xrightarrow{a} \mathbf{1} \cdot (b \cdot a)^*$$

Congruence & algebraic properties

- Congruence: if p $\sim_f$ q, then p* $\sim_f$ q*
- Algebraic properties of iteration:

$$\begin{array}{lrl} (i) & p^* & \sim_f p \cdot p^* + \mathbf{1} \\ (ii) & p^* & \sim_f (p + \mathbf{1})^* \\ (iii) & (p+q)^* & \sim_f p^* \cdot (q \cdot (p+q)^* + \mathbf{1}) \end{array}$$

- Any BPA* process p generates a finite-state lts-f. (ONLY)
- The proof, by structural induction on p, shows that s-size(p) is a finite number, where s-size(p) is an upper-bound on the number of reachable states

$$\begin{array}{rlrl} \textit{s-size}(\mathbf{0}) & = 1 & \textit{s-size}(\mathbf{1}) & = 1 \\ \textit{s-size}(\mu) & = 2 & \textit{s-size}(p_1 + p_2) & = \textit{s-size}(p_1) + \textit{s-size}(p_2) + 1 \\ \textit{s-size}(p^*) & = \textit{s-size}(p) + 1 & \textit{s-size}(p_1 \cdot p_2) & = \textit{s-size}(p_1) + \textit{s-size}(p_2) \end{array}$$

All Finite state lts-f?

- We have seen that only finite-state lts-f are representable by BPA*.
- Can any finite-state lts-f be represented, up to $\sim f$, by a suitable BPA* process? NO (see next slide)
- However, any finite-state lts-f can be represented, up to f-trace equivalence, by a suitable BPA* process. This result is rather expected, as BPA* is the language of regular expressions!

Theorem 5.5. *For any any* $p \in \mathscr{P}_{BPA^*}$, $Tr_f(p) = \mathscr{L}[p]$.

Not all, up to f-bisimilarity

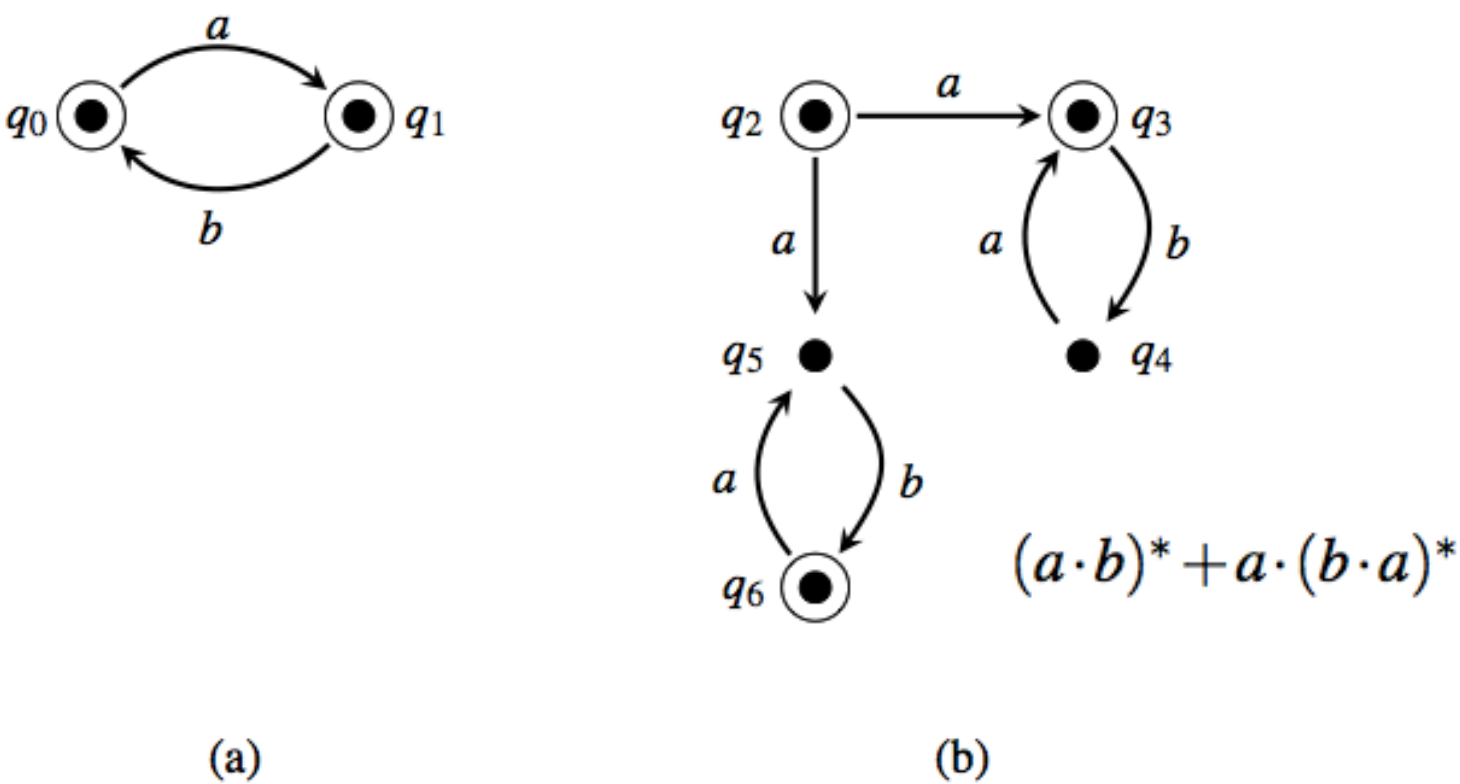


Fig. 5.4 A lts-f not representable in BPA*, up to $\sim_f$, in (a); and one of its f-trace equivalent lts-f's in (b).

BPA: finite BPA with recursion

- Syntax $p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p+p \mid p \cdot p \mid C$
- SOS rules

$$\frac{p\downarrow}{C\downarrow}\ C \stackrel{def}{=} p \qquad (\text{Cons})\ \frac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}\ C \stackrel{def}{=} p$$

- Example: the BPA process for the language ww^R

$$S \stackrel{def}{=} a \cdot S \cdot a + b \cdot S \cdot b + \mathbf{1}$$

- Trace abba: $S \xrightarrow{a} \mathbf{1} \cdot S \cdot a \xrightarrow{b} \mathbf{1} \cdot S \cdot b \cdot a \xrightarrow{b} \mathbf{1} \cdot a \xrightarrow{a} \mathbf{1}$

From contex-free grammars (in Greibach form) to BPA processes

- For any context-free grammar G, there exists an equivalent grammar G' in Greibach form: all of its productions are of the form $B \rightarrow \alpha\beta$, where α is a terminal symbol and β a sequence (possibly empty) of nonterminal symbols

Given a Greibach normal form grammar $G = (N, T, S, P)$, it is enough to define a BPA constant V in correspondence of each nonterminal $V \in N$; if nonterminal V has productions $V \to a_1\gamma_1 \ldots V \to a_k\gamma_k$, then constant V has k summands, each one corresponding to the obvious translation of a sequence $a_i\gamma_i = a_iX_1 \ldots X_{n_i}$ into the BPA process $a_i \cdot X_1 \cdot \ldots \cdot X_{n_i}$ (if $V \to \varepsilon$, then the corresponding BPA process constant V has a summand $\mathbf{1}$ in its body). The process p such that $L(G) = Tr_f(p)$ is simply constant S, as S is the initial nonterminal symbol of G.

- For instance: $G = (\{S\}, \{a, b\}, S, \{S \to aSa, S \to bSb, S \to \varepsilon\})$

$$S \stackrel{def}{=} a \cdot S \cdot a + b \cdot S \cdot b + \mathbf{1}$$

From BPA processes to context-free grammars (Example)

$$C \stackrel{def}{=} (a+b)\cdot D\cdot c+\mathbf{0}$$
$$D \stackrel{def}{=} a\cdot(\mathbf{0}+\mathbf{1})\cdot C+\mathbf{1}$$

By resorting to the algebraic laws discussed in Exercises 5.19, 5.20 and 5.21, we are able to define f-trace equivalent process constants C' and D', respectively. Of particular interest are the two distributivity laws: $(p+q)\cdot r \sim_f p\cdot r+q\cdot r$ that holds for f-bisimilarity, and $r\cdot(p+q) =_{trf} r\cdot p+r\cdot q$ that holds for f-trace equivalence. By using these laws, we can equivalently write:

$$C' \stackrel{def}{=} a\cdot D'\cdot c+b\cdot D'\cdot c+\mathbf{0}$$
$$D' \stackrel{def}{=} a\cdot\mathbf{0}\cdot C'+a\cdot\mathbf{1}\cdot C'+\mathbf{1}$$

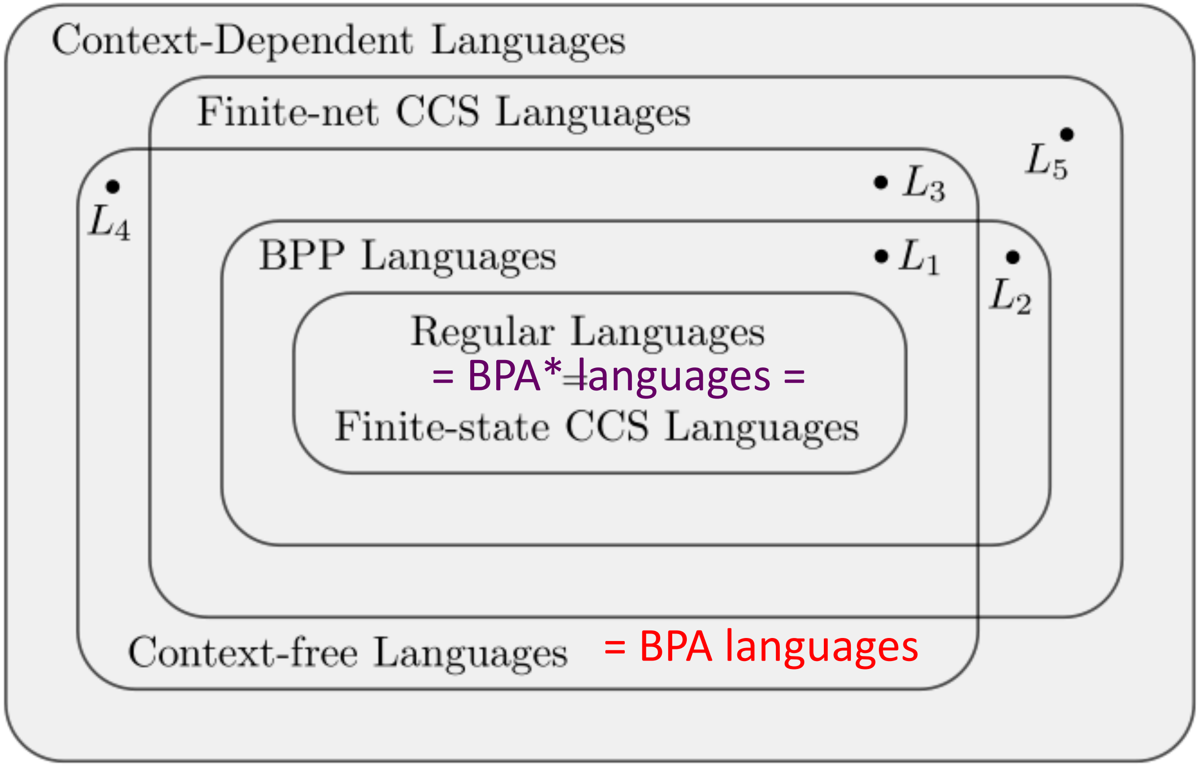
By using the law $p\cdot\mathbf{0} =_{trf} \mathbf{0}$, $\mathbf{0}\cdot p \sim_f \mathbf{0}$, $p\cdot\mathbf{1} \sim_f p$ and $p+\mathbf{0} \sim_f p$, we can then obtain the f-trace equivalent forms:

$$C'' \stackrel{def}{=} a\cdot D''\cdot c+b\cdot D''\cdot c$$
$$D'' \stackrel{def}{=} a\cdot C''+\mathbf{1}$$

$$S \to aAc \quad S \to bAc$$
$$A \to aS \quad A \to \varepsilon$$

Summing up …

- BPA languages are exactly context-free languages!



Counter

- A counter can be easily represented in BPA with two constants only! (and no restriction!)

$$BC \stackrel{def}{=} zero \cdot BC + inc \cdot (S \cdot BC)$$
$$S \stackrel{def}{=} dec + inc \cdot (S \cdot S)$$

Decidability issues

- Trace equivalence is undecidable for BPA
- Bisimulation equivalence is decidable for BPA: Unfortunately, the best known algorithm is doubly exponential for the general case, even if efficient, polynomial algorithms are available for the normed case (i.e., for those BPA processes that can always terminate successfully)
- The problem of deciding weak bisimilarity for BPA is open.

Encoding iteration with recursion

Exercise 5.29. (Encoding the iteration operator) Let BPA^{+*} denote the language BPA enriched with the iteration construct. Show that BPA^{+*} can be implemented into BPA by showing a simple encoding $[\![-]\!]$ of p^* by means of a recursively defined constant. (*Hint:* Look at the algebraic property $p^* \sim_f p \cdot p^* + \mathbf{1}$.) $\square$

- p* can be implemented by a recursive constant A_p as follows: $A_p = p \cdot A_p + 1$

PA

- Syntax: adding to BPA asynchronous parallel composition, as we did for BPP:

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p+p \mid p \cdot p \mid p|p \mid C$$

$$\frac{p\downarrow \quad q\downarrow}{(p|q)\downarrow} \qquad (\text{Par}_1)\ \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q} \qquad (\text{Par}_2)\ \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

- Algebraic properties:

$$\begin{aligned} p|(q|r) &\sim_f (p|q)|r \\ p|q &\sim_f q|p \\ p|\mathbf{1} &\sim_f p \\ p|\mathbf{0} &\sim_f p \cdot \mathbf{0} \end{aligned}$$

What about PA?

- PA is a syntactic extension of BPA, because asynchronous parallel composition is added to PA, and also of BPP, as sequential composition is a generalization of action prefixing. Not surprisingly, PA is strictly more expressive than both BPA and BPP. It is incomparable w.r.t. finite-net CCS.
- PA is not a Turing-complete language because it was proved that the reachability problem is still decidable for PA.
- The problem of deciding bisimilarity for PA is open, though for normed PA processes a positive decidability result already exists. Checking weak bisimilarity for PA is undecidable.

PAER

- A natural extension of PA is PAER, obtained by adding the communication capability to the parallel operator and an external operator of restriction, as we did when extending BPP to finite-net CCS.

$$
\begin{aligned}
p &::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p+p \mid p\cdot p \mid p|p \mid C \\
q &::= p \mid (\nu a)q
\end{aligned}
$$

$$
(\text{Com})\ \frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\overline{\alpha}} q'}{p|q \xrightarrow{\tau} p'|'q} \qquad \frac{p\downarrow}{(\nu a)p\downarrow} \qquad (\text{Res})\ \frac{p \xrightarrow{\mu} p'}{(\nu a)p \xrightarrow{\mu} (\nu a)p'}\ \mu \neq a, \bar{a}
$$

PAER is Turing-complete

- PAER is a Turing-complete formalism, as any counter machine *M* can be encoded in PAER. It is enough to adapt the construction described in Section 3.5.2 for finitary CCS
- The crucial components are the counters, that can be modeled with BPA processes, so that the resulting process

$$CM_{M(v_1,v_2,v_3)} \stackrel{def}{=} (\nu L)(P_1 \mid \ldots \mid P_m \mid R_1 \mid R_2 \mid R_3 \mid B_{(v_1,v_2,v_3)})$$

is a PAER process indeed! Only external restrictions are used, as the counters do not use restriction at all, rather sequential composition.

Expressiveness

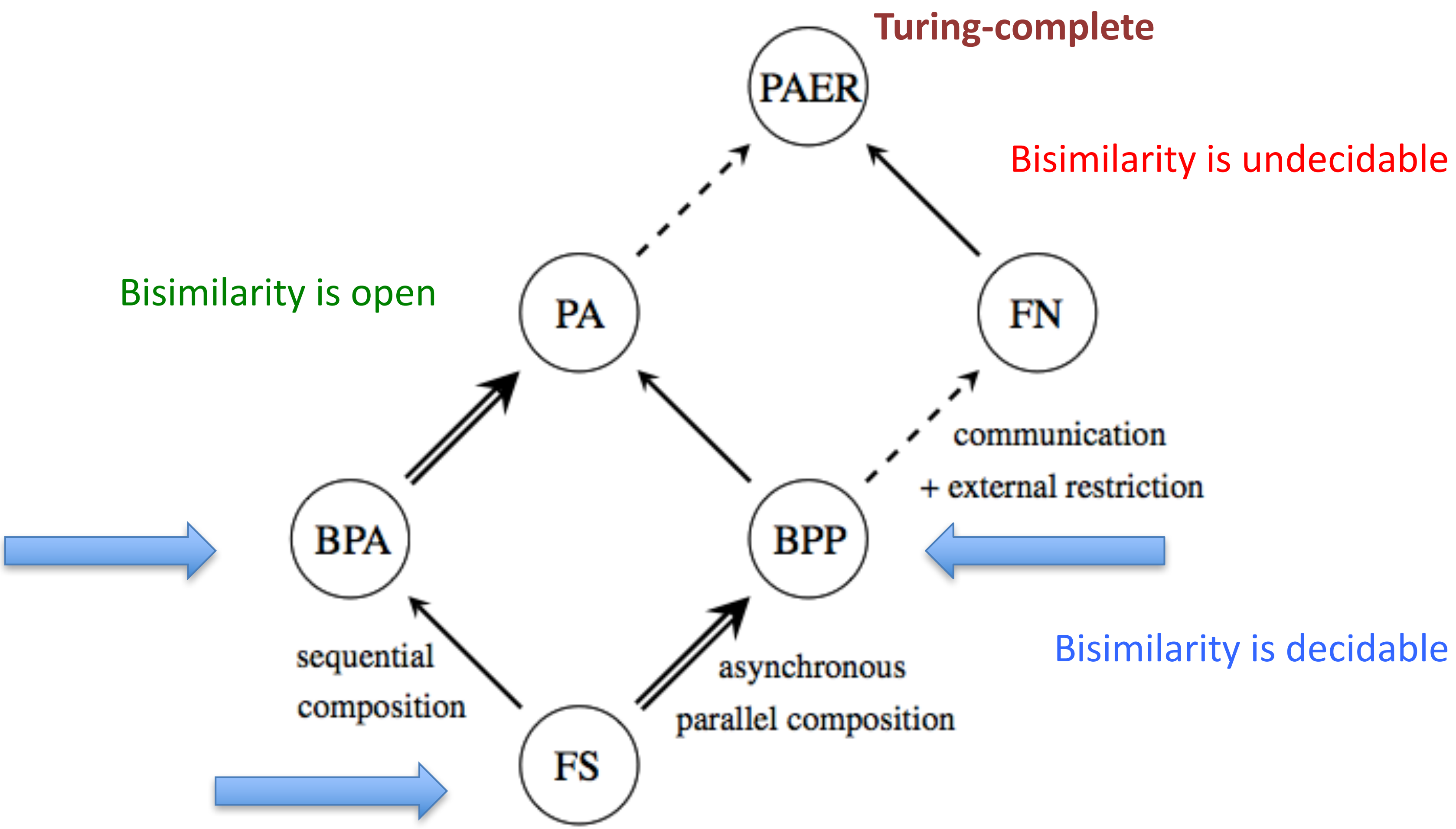


Fig. 5.5 Syntactic and semantic classification of some calculi (FS stands for finite-state CCS, FN for finite-net CCS).

CCS^{seq} syntax

- CCS^{seq}: extension of CCS (in compact form) with sequential composition

$$p ::= \Sigma_{j \in J} \mu_j . p_j \;\Big|\; p | p \;\Big|\; p \cdot p \;\Big|\; (\nu a) p \;\Big|\; C$$

- Note that it is not allowed to write terms of the form *a*.0 + 0, i.e., 0 is not allowed as a summand; this is not strictly necessary, but it simplifies the correctness proof.

CCS^{seq} final states

- Final state is 0 (not 1)

$$\frac{}{\mathbf{0}\downarrow} \qquad \frac{p\downarrow \quad q\downarrow}{(p\cdot q)\downarrow} \qquad \frac{p\downarrow \quad q\downarrow}{(p|q)\downarrow} \qquad \frac{p\downarrow}{(\nu a)p\downarrow} \qquad \frac{p\downarrow}{C\downarrow}\ C\stackrel{def}{=}p$$

- There is no basic non-final state. An example of a deadlock non-final state is (νa)a.0
- Note that a term of the form *p* + *q* cannot be final: by syntactic construction, both *p* and *q* should start with a prefix, hence they are not final.

CCS^{seq} semantics

- Usual SOS rules for CCS with additionally:

$$(\mathrm{Seq}_1)\ \frac{p \xrightarrow{\mu} p'}{p \cdot q \xrightarrow{\mu} p' \cdot q} \qquad (\mathrm{Seq}_2)\ \frac{p \downarrow \quad q \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} q'}$$

Exercise 5.31. Prove that, for any $p \in \mathscr{P}_{CCS^{seq}}$, if $p \downarrow$, then $p \not\rightarrow$. (*Hint:* by induction on the proof of $p \downarrow$.) Note that this fact is not true for BPA and related languages; e.g., the BPA process $p = a + \mathbf{1}$ is such that $p \downarrow$ and $p \xrightarrow{a}$. $\square$

Algebraic properties & congruence

$$
\begin{array}{ll}
(i) & p \cdot (q \cdot r) \sim_f (p \cdot q) \cdot r \\
(ii) & p \cdot q \sim_f q \quad \text{if } p\downarrow \\
(iii) & q \cdot p \sim_f q \quad \text{if } p\downarrow \\
(iv) & (p+q) \cdot r \sim_f p \cdot r + q \cdot r \\
(v) & p \,|\, q \sim_f q \quad \text{if } p\downarrow \\
(vi) & (\nu a)p \sim_f p \quad \text{if } a \notin fn(p)
\end{array}
$$

1) If $p \sim_f q$, then $p \cdot r \sim_f q \cdot r$ and $r \cdot p \sim_f r \cdot q$, for all $r \in \mathscr{P}_{CCS^{seq}}$,
2) If $p \approx_f q$, then $p \cdot r \approx_f q \cdot r$ and $r \cdot p \approx_f r \cdot q$, for all $r \in \mathscr{P}_{CCS^{seq}}$.

1) If $p \sim_f q$, then $p \,|\, r \sim_f q \,|\, r$ for all $r \in \mathscr{P}_{CCS^{seq}}$,
2) If $p \approx_f q$, then $p \,|\, r \approx_f q \,|\, r$ for all $r \in \mathscr{P}_{CCS^{seq}}$,
3) If $p \sim_f q$, then $(\nu a)p \sim_f (\nu a)q$ for all $a \in \mathscr{L}$,
4) If $p \approx_f q$, then $(\nu a)p \approx_f (\nu a)q$ for all $a \in \mathscr{L}$.

Derived operator: encoding CCS^{seq} into CCS up to $\approx_f$

- The encoding $[\![-]\!]$ of CCS^{seq} into CCS, up to weak f-bisimilarity $\approx_f$ is homomorphic for all operators, except sequential composition:

$$[\![p \cdot q]\!] = (\nu d)([\![p]\!]^e_d \,|\, \overline{d}.[\![q]\!]) \quad d, e \notin fn(p \cdot q) \cup bn(p \cdot q)$$

- This encoding uses an auxiliary encoding $[\![-]\!]^e_d$ parametrized on two new, distinct names *d* and *e*, such that *d* is a free name and *e* a bound name. The intuition is that whenever *p* reaches a final state *p'*, then $[\![p']\!]^e_d$ can perform *d* as its last action and then deadlocks; this is made clear by the rule: $[\![\mathbf{0}]\!]^e_d = d.\mathbf{0}$

Auxiliary encoding of parallel composition

- The auxiliary encoding of parallel composition reveals the need for the additional new bound name *e*

$$[\![p_1 \mid p_2]\!]^e_d = (\nu e)(([\![p_1]\!]^d_e \mid [\![p_2]\!]^d_e) \mid \bar{e}.\bar{e}.d.\mathbf{0})$$

- On the one hand, $[\![p1]\!]^d_e$ will possibly terminate its execution by performing *e* (using *d* as an auxiliary bound name) if *p*1 terminates successfully; similarly, $[\![p2]\!]^d_e$. On the other hand, by restricting on action *e*, two synchronizations with the two occurrences of action *e* in the third component are to be performed, with the effect of activating subprocess *d*.**0**: hence, the last action that is performed by $[\![p1|p2]\!]^e_d$ is *d* indeed, provided that both components terminate successfully.

Auxiliary encoding of sequential composition

A similar inversion of the roles between *d* and *e* is present also in the auxiliary encoding of sequential composition:

$$[\![p \cdot q]\!]_d^e = (\nu e)([\![p]\!]_e^d \mid \overline{e}.[\![q]\!]_d^e)$$

Where in $[\![p \cdot q]\!]^e_d$ action *d* is free and *e* is bound, while in $[\![p]\!]^d_e$ action *e* is free and *d* is bound. Process $[\![p]\!]^d_e$ will possibly end its execution by performing *e*, to be synchronized with ‘$e.[\![q]\!]^e_d$ because of the restriction on the auxiliary *e*. Then $[\![q]\!]^e_d$ will possibly end by performing *d*, as required by $[\![p \cdot q]\!]^e_d$.

- Note that two new names, *d* and *e*, are enough: the two names give rise to alternated restrictions so that no confusion can be generated (no capture of free names).

Summing up …

$$[\![\mathbf{0}]\!] = \mathbf{0} \qquad [\![\mu.p]\!] = \mu.[\![p]\!] \qquad [\![p_1 + p_2]\!] = [\![p_1]\!] + [\![p_2]\!] \qquad [\![p_1 \,|\, p_2]\!] = [\![p_1]\!] \,|\, [\![p_2]\!]$$

$$[\![(\nu a)p]\!] = (\nu a)[\![p]\!] \qquad [\![A]\!] = A' \quad \text{where } A' \stackrel{def}{=} [\![p]\!] \text{ if } A \stackrel{def}{=} p$$

$$[\![p \cdot q]\!] = (\nu d)([\![p]\!]_d^e \,|\, \overline{d}.[\![q]\!]) \quad d, e \notin fn(p \cdot q) \cup bn(p \cdot q)$$

$$[\![\mathbf{0}]\!]_d^e = d.\mathbf{0} \qquad [\![\mu.p]\!]_d^e = \mu.[\![p]\!]_d^e \qquad [\![p_1 + p_2]\!]_d^e = [\![p_1]\!]_d^e + [\![p_2]\!]_d^e$$

$$[\![p_1 \,|\, p_2]\!]_d^e = (\nu e)(([\![p_1]\!]_e^d \,|\, [\![p_2]\!]_e^d) \,|\, \overline{e}.\overline{e}.d.\mathbf{0})$$

$$[\![(\nu a)p]\!]_d^e = (\nu a)[\![p]\!]_d^e \qquad [\![A]\!]_d^e = A_{ed} \quad \text{where } A_{ed} \stackrel{def}{=} [\![p]\!]_d^e \text{ if } A \stackrel{def}{=} p$$

$$[\![p \cdot q]\!]_d^e = (\nu e)([\![p]\!]_e^d \,|\, \overline{e}.[\![q]\!]_d^e)$$

Table 5.8 Encoding CCS^{seq} into CCS.

The encoding is correct

- Theorem: p and $[\![p]\!]$ are weakly f-bisimilar
- Very technical proof: the natural candidate relation

$$R = \{(p, [\![p]\!]) \mid p \in \mathscr{P}_{CCS^{seq}}\}$$

is not a weak f-bisimulation up to $\approx_f$

- (See the book, if interested, for the details: a good example on how to prove a compiler correct)

CSP Multi-party synchronization

- While in CCS synchronization is point-to-point (strictly binary), in CSP communication is multi-party, i.e., one single synchronization step may involve many different sequential processes.
- In CSP the set *Act* of actions is not partitioned into the subsets of input actions and output actions (co-actions), as in CCS; so actions have no type.
- The parallel composition of two processes, say p and q, is parametrized by a set of actions $A \subseteq Act \setminus \{tau\}$, called *synchronization set*, and take the following syntactic form: $p \,||_A\, q$.

CSP operational rules

$$(\mathrm{Csp}_1)\ \frac{p \xrightarrow{\mu} p'}{p \parallel_A q \xrightarrow{\mu} p' \parallel_A q}\ \mu \notin A \qquad (\mathrm{Csp}_2)\ \frac{q \xrightarrow{\mu} q'}{p \parallel_A q \xrightarrow{\mu} p \parallel_A q'}\ \mu \notin A$$

$$(\mathrm{Csp}_3)\ \frac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \parallel_A q \xrightarrow{a} p' \parallel_A q'}\ a \in A$$

- A synchronization between *p* and *q* can occur only if both are able to perform the very same action $a \in A$; the effect is that $p \,||_A\, q$ also performs action *a*; thus, contrary to CCS, the result of a synchronization is observable and it can be used for further synchronization with other parallel subprocesses.
- Within $p \,||_A\, q$, processes *p* and *q* cannot perform asynchronously any action belonging to the synchronization set *A*; on the contrary, *p* and *q* cannot synchronize on actions not belonging to *A*.

How may multi-party synch happen?

A simple example may help clarifying how multi-party synchronization can take place in CSP. Let us consider process $(a.\mathbf{0} \parallel_{\{a\}} a.\mathbf{0}) \parallel_{\{a\}} a.\mathbf{0}$. The following proof tree shows how to derive transition $(a.\mathbf{0} \parallel_{\{a\}} a.\mathbf{0}) \parallel_{\{a\}} a.\mathbf{0} \xrightarrow{a} (\mathbf{0} \parallel_{\{a\}} \mathbf{0}) \parallel_{\{a\}} \mathbf{0}$:

$$
\dfrac{\dfrac{\dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}}(\text{Pref}) \qquad \dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}}(\text{Pref})}{a.\mathbf{0} \parallel_{\{a\}} a.\mathbf{0} \xrightarrow{a} \mathbf{0} \parallel_{\{a\}} \mathbf{0}}(\text{Csp}_3) \qquad \dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}}(\text{Pref})}{(a.\mathbf{0} \parallel_{\{a\}} a.\mathbf{0}) \parallel_{\{a\}} a.\mathbf{0} \xrightarrow{a} (\mathbf{0} \parallel_{\{a\}} \mathbf{0}) \parallel_{\{a\}} \mathbf{0}}(\text{Csp}_3)
$$

- Of course, as the result of a synchronization is observable, it is necessary to confine the visibility of names by means of the hiding operator (i*a*)*p*. A process (i*a*)(*p*1 $||_{\{a\}}$ *p*2) $||_{\{a\}}$ *q* is such that *p*1 and *p*2 can synchronize over action *a*, which is then turned into tau by hiding, so that *q* cannot interact with them over action *a*.

Is CSP parallel composition encodable into CCS?

- The answer to this question is negative:
- CSP is powerful enough to solve the dining philosopher problem with an algorithm which is fully distributed, symmetric, deadlock-free and divergence-free, while this is impossible for CCS!
- If we want to encode CSP parallel composition into CCS, we have to extend CCS capabilities.
- Multi-CCS includes one additional operator, called *strong prefixing*, that allows for the implementation of atomic transactions. In this setting, a multi-party synchronization can be implemented as an atomic sequence of binary CCS synchronizations.