Graphical Models of Separation Logic

Ian Wehrman, Tony Hoare, Peter O'Hearn

Outlook

- Present a new model of programs and assertions for a variety of languages.
- Use model for language-independent reasoning.

Program logics

- Logics tailored for program correctness:
  - programming + assertion languages,
  - program + assertion semantics,
  - axioms and inference rules.

Hoare logic

- Program logic for while-loop languages.
- Hoare triple: P {C} Q
  - C is a program,
  - P and Q are assertions about program state.
- Informal meaning:
  - if C is run in a P-state, then (if it halts) it halts in a Q-state.

Semantics of P {C} Q

- Assertion semantics:
  - $\langle\!\langle - \rangle\!\rangle : \text{Assertions} \to \mathcal{P}(\text{States})$
- Program semantics:
  - $[\![-]\!] : \text{Programs} \to (\text{States} \to \mathcal{P}(\text{States}))$
- Triple semantics:
  - $\forall s \in \langle\!\langle P \rangle\!\rangle \,.\, [\![C]\!](s) \subseteq \langle\!\langle Q \rangle\!\rangle$

Proving P {C} Q

- Axioms:
  - e.g., P[E/x] {x := E} P

    y=3 {x := y} x=3
- Inference rules:
  - e.g., P {C} R and R {C'} Q implies P {C;C'} Q.

    if y=3 {x := y} x=3 and x=3 {z := x} z=3

    then y=3 {x:=y ; z:=x} z=3

Soundness

- Axioms are true:
  - $\forall s \in \langle\!\langle P[E/x] \rangle\!\rangle \;.\; [\![x{:=}E]\!](s) \subseteq \langle\!\langle P \rangle\!\rangle$
- Inference rules preserve truth:
  - If $\forall s \in \langle\!\langle P \rangle\!\rangle \;.\; [\![C]\!](s) \subseteq \langle\!\langle R \rangle\!\rangle$, and
  - $\forall s \in \langle\!\langle R \rangle\!\rangle \;.\; [\![C']\!](s) \subseteq \langle\!\langle Q \rangle\!\rangle$
  - then $\forall s \in \langle\!\langle P \rangle\!\rangle \;.\; [\![C;C']\!](s) \subseteq \langle\!\langle Q \rangle\!\rangle$

Separation logic

- Program logic for C programs (pointers)
  - different program state: vars+heap
  - different assertion language: P*Q
  - different semantic functions: 《-》 and ⟦-⟧
  - different axioms
  - same inference rules + extras

Tony has a dream

- ...a unified theory of programming.
- Most languages share basic constructs: e.g., sequentiality and concurrency
    - Reasoning about general features should be language-independent
    - Reasoning about specific features should be language-specific

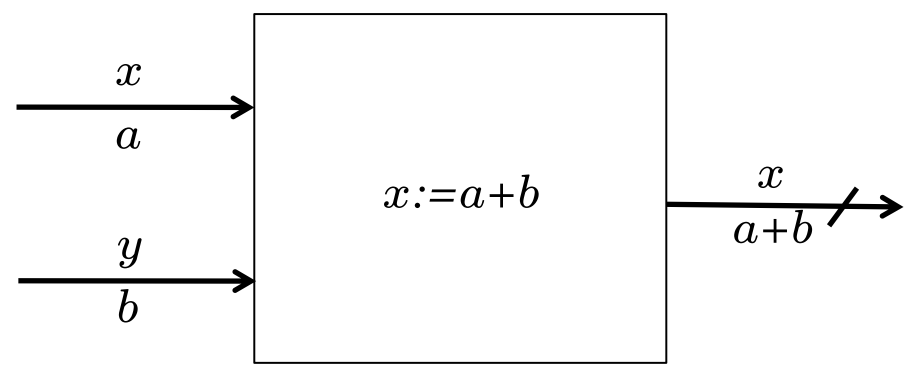
Today

- Toward language-independent reasoning:
  - present a very general model of all kinds of programs and assertions
  - characterize sequentiality and concurrency
  - give semantics to triples in this model
  - show that inference rules still hold

A very general model

- Sets of labeled directed graphs
- Graph represents a program execution:
  - nodes – events that occur during execution
  - edges – dependency between events
  - labels – information flow

Simple assignment



x := x + y

Assertions as programs

- Use same model for assertions as programs
- Assertions as underspecified programs:
  - e.g. "x=2 \/ y=3" any execution in which either the last write to x is 2, or last to y is 3.

Usage

- To use this for <u>your</u> language, provide semantic functions:
  - $\langle\!\langle - \rangle\!\rangle$: Assertions → $\mathcal{P}$(Graphs)
  - $[\![-]\!]$: Programs → $\mathcal{P}$(Graphs)
- Today, ignore languages, just deal with <u>arbitrary</u> sets of graphs P.

Dependency

- To define sequentiality and concurrency, consider dependency between events.
- $p \rightarrow q$ means "event q depends on event p"
    - Might describe control flow, data flow, etc.

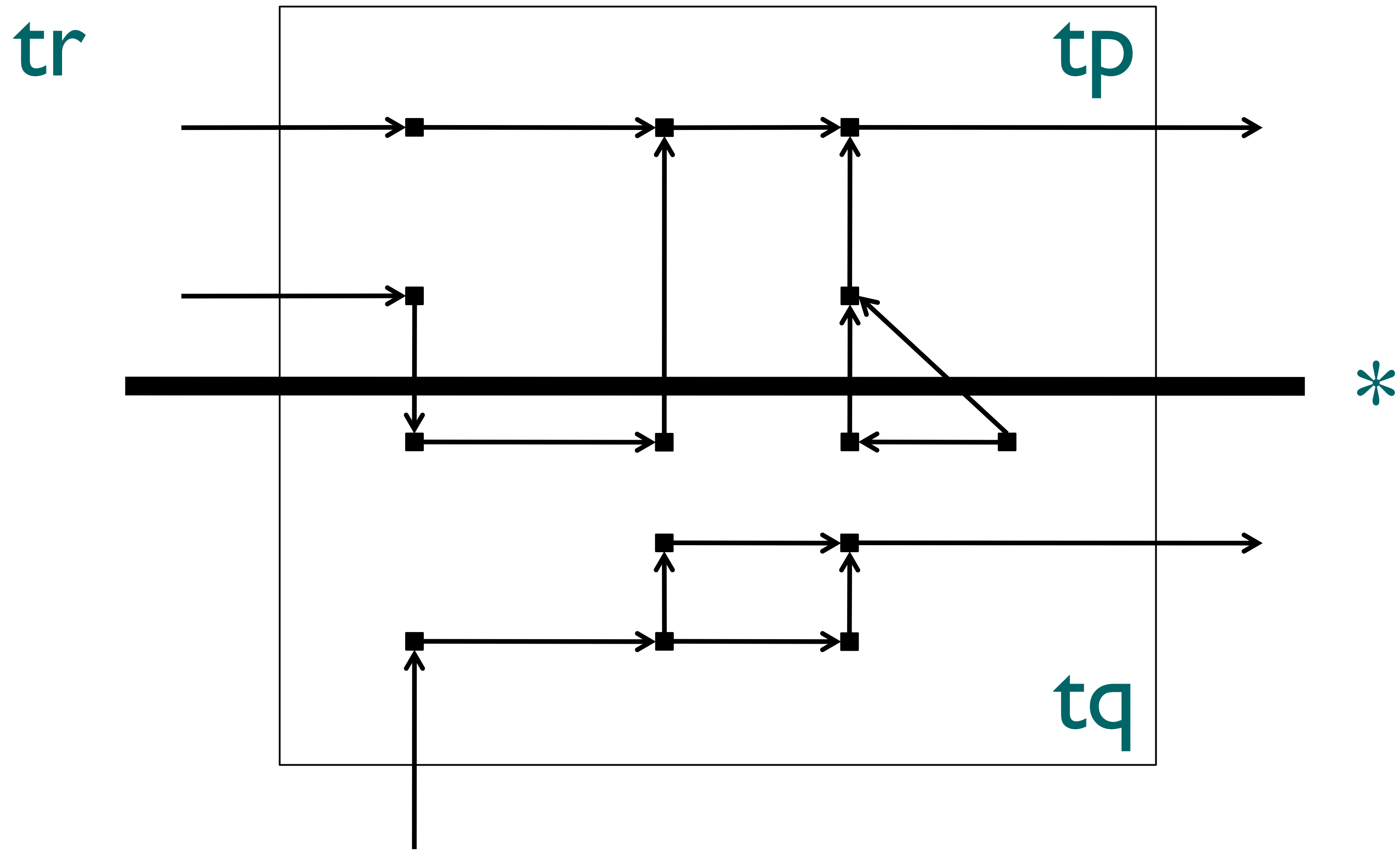
Traces

- *trace*: subset of events from an execution
- Represents execution of part of a program
- Lift dependency to traces:
    - $tp \rightarrow tq$ means $\exists p \in tp, q \in tq$ with $p \rightarrow q$

Concurrency

- A trace can be separated into concurrent parts by partitioning its events:
  - Write $tp*tq$ for *concurrent composition* of traces.
  - $tr=tp*tq$ iff $tr=tp\cup tq$ and $tp\cap tq=\varnothing$.
  - $P*Q = \{tr \mid \exists tp\in P, tq\in Q \;.\; tr=tp*tq\}$.

Concurrency

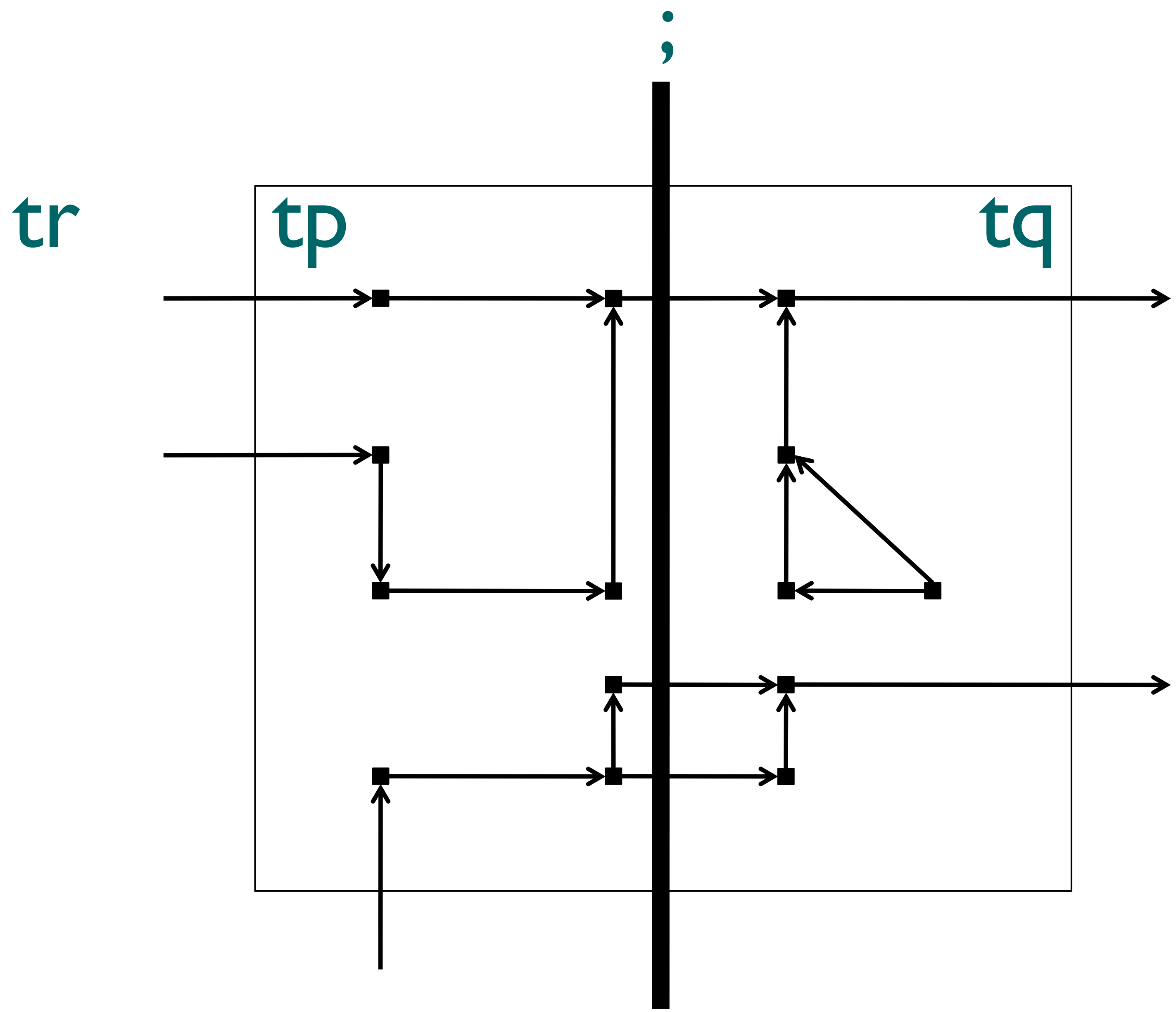


tr = tp*tq

Sequentiality

- A trace can be separated sequentially by partitioning and respecting dependency:
  - Write tp;tq for *sequential composition*
  - tr=tp;tq iff tr=tp*tq and $\neg$(tq$\rightarrow$tp)
  - P;Q = {tr | $\exists$tp$\in$P, tq$\in$Q . tr=tp;tq}.

Sequentiality



tr = tp;tq

Other constructions

- skip = $\{\varnothing\}$
- false = $\varnothing$
- Disjunction: $P \vee Q = P \cup Q$
- Conjunction: $P \wedge Q = P \cap Q$

Refinement

- $P \models Q$ means $P \subseteq Q$
  - program P *refines* program Q
  - assertion P *semantically entails* assertion Q
  - program P *satisfies* assertion Q
- e.g., $P \wedge Q \models P$, but <u>not</u> $P * Q \models P$

Algebra

- skip is unit, false is zero
- (;) is associative, $\vee$-distributive, monotonic
  - $P \models Q$ implies $P;R \models Q;R$
- (;) satisfies Kleene laws:
  - P^* is least fixpoint of $\lambda X.(\mathit{skip} \vee (X;P))$

Algebra

- (*) satisfies all properties of (;), plus commutativity
  - $P;Q \models P*Q$
- Exchange Law relates (;) and (*):
  - $(P*Q);(P'*Q') \models (P;P')*(Q;Q')$

Hoare triples

- New semantics: $P \{Q\} R$ means $(P;Q) \vDash R$.
  - any trace $tp;tq$ that starts with $tp \in P$ and then does $tq \in Q$ must also be in R

Inference rules

- if $P\ \{Q\}\ R$ and $P\ \{Q\}\ R'$ then $P\ \{Q\}\ R \wedge R'$
- if $P\ \{Q\}\ R$ and $P'\ \{Q\}\ R$ then $P \vee P'\ \{Q\}\ R$
- if $P\ \{Q\}\ S$ and $S\ \{Q'\}\ R$ then $P\ \{Q;Q'\}\ R$

Sequential Proof

P;(Q;Q')

$\models$ { associativity of (;) }

(P;Q);Q'

$\models$ { first assumption, P;Q $\models$ S, and monotonicity of (;) }

S;Q'

$\models$ { second assumption S;Q' $\models$ R }

R.

Separation Logic

- if P {Q} R and P’ {Q’} R’ then P*P’ {Q*Q’} R*R’
    - Disjoint concurrency
- if P {Q} R then P*F {Q} R*F
    - Frame rule

Frame Proof

$(P*F);Q$

$\models$ { commutativity of $(*)$ }

$(F*P);Q$

$\models$ { Exchange theorem, with $(Q*skip) = Q$ }

$F*(P;Q)$

$\models$ { assumption $P;Q \models R$, monotonicity of $(*)$ }

$R*F$.

Beyond

- Graphs also model logic for non-disjoint concurrency:
  - rely/guarantee
- But healthiness conditions are needed:
  - transitivity and acyclity

Conclusion

- Presented new, general model for programs, assertions and program logics
- Characterized concurrency and sequentiality
- Language-independent validation of inference rules of program logics

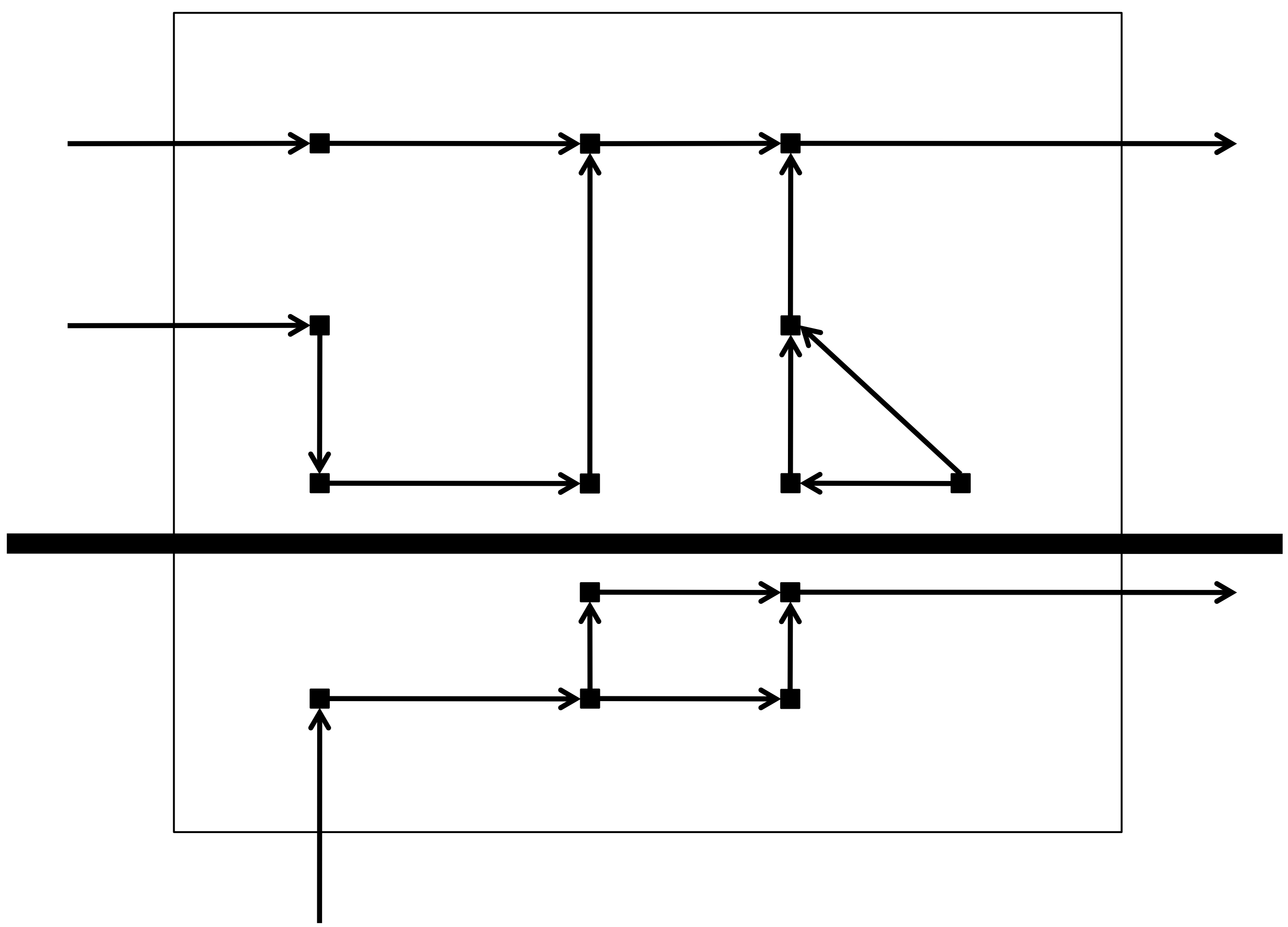
References

- Hoare, Wehrman, O'Hearn. *Graphical Models of Separation Logic*. Marktoberdorf Summer School 2008.
- Wehrman, Hoare, O'Hearn. *Graphical Models of Separation Logic*. Information Processing Letters, 2009.
- Hoare, Möller, Struth, Wehrman. *Concurrent Kleene Algebra*. CONCUR 2009.
- Hoare, Möller, Struth, Wehrman. *Foundations of Concurrent Kleene Algebra*. REL/MICS 2009.

Parallelism

- Write tp||tq for *parallel composition*
    - tr=tp||tq iff tr=tp;tq and tr=tq;tp
- P||Q is parallel composition of P and Q
    - P||Q = (U tp∈P, tq∈Q : tp||tq)

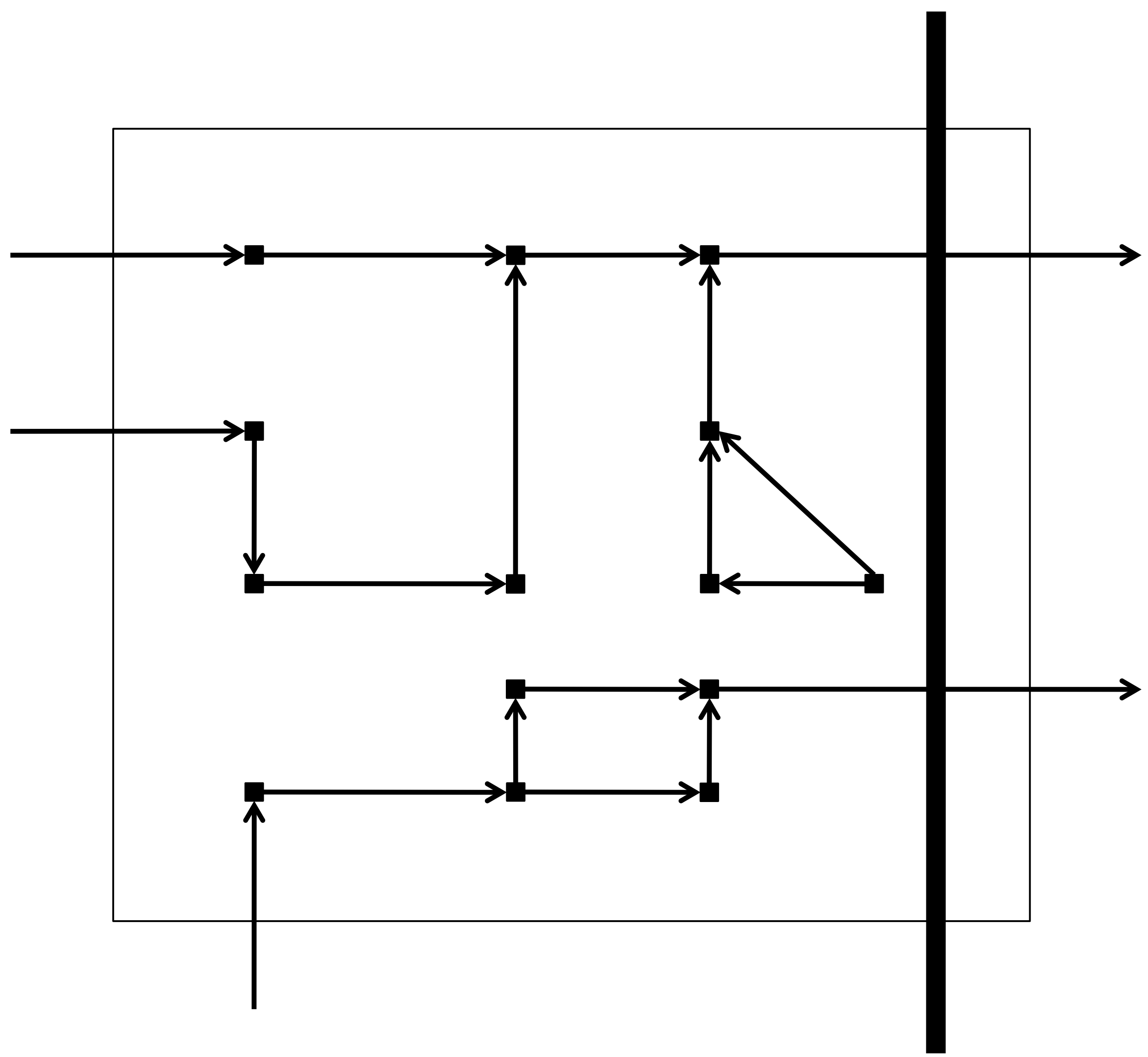
Parallelism



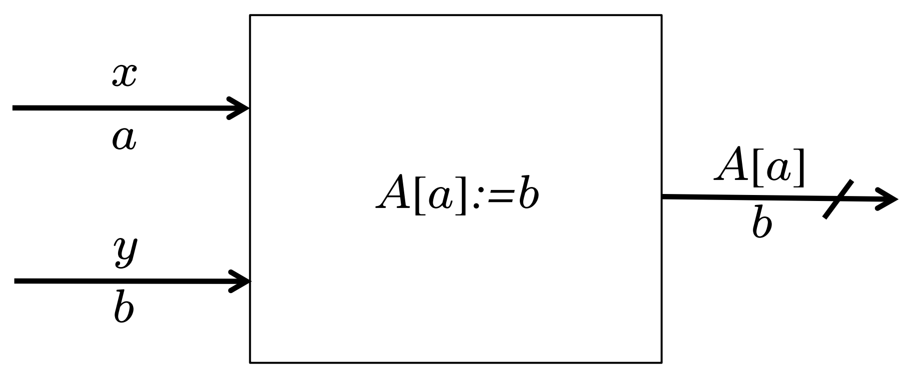
Choice

- Write $tp[]tq$ for *nondeterministic choice*
    - $tr=tp[]tq$ iff $tp=tp \cup tq$ and ($tp=\varnothing$ or $tq=\varnothing$)
- $P[]Q$ is choice between P and Q
    - $P[]Q = (\cup\ tp \in P, tq \in Q : tp[]tq)$

Choice

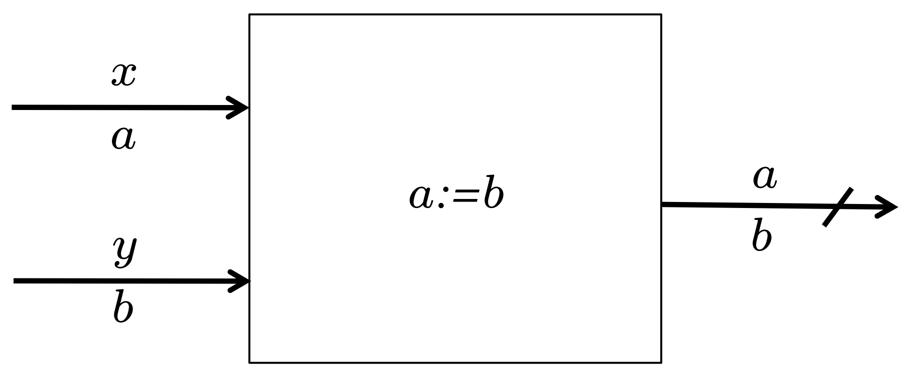


Array Assignment



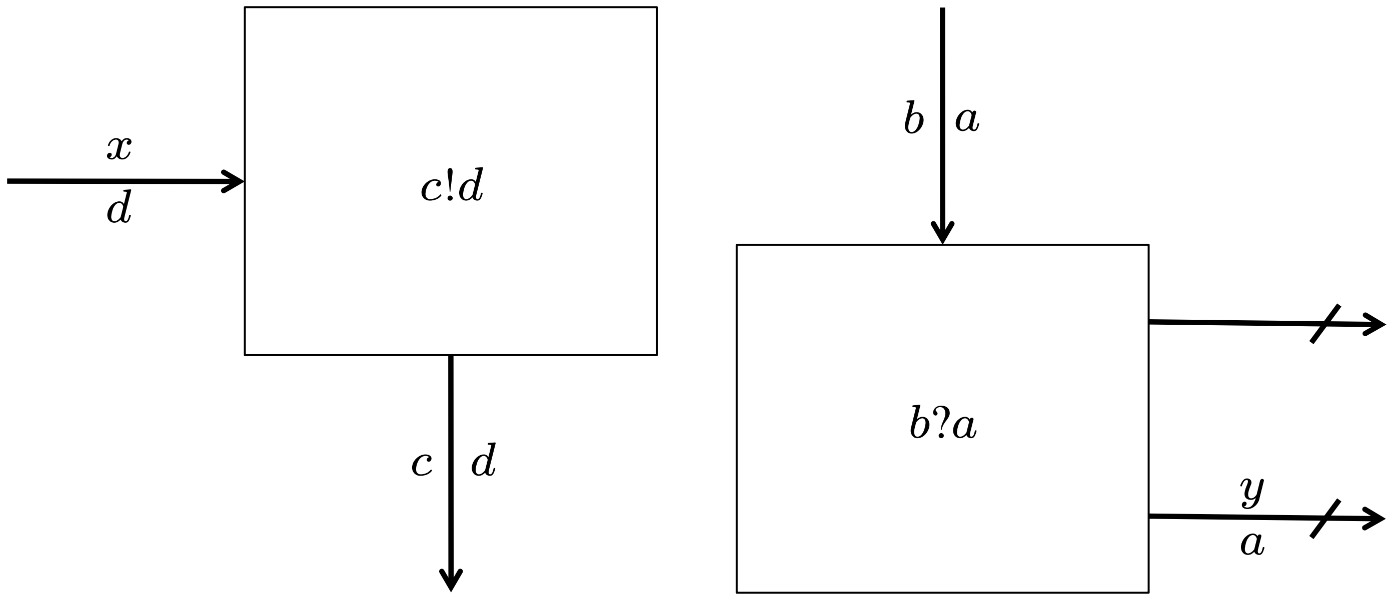
$A[x] = y$

Indirect Assignment



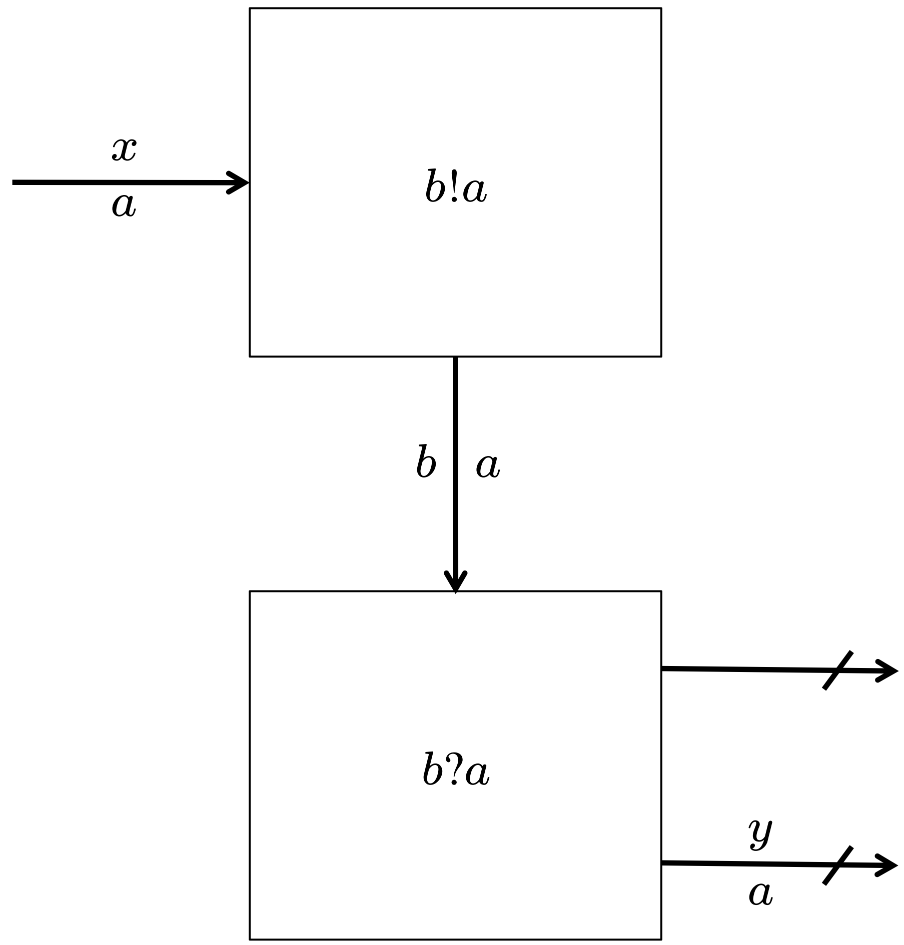
x* = y

CSP Interleaving



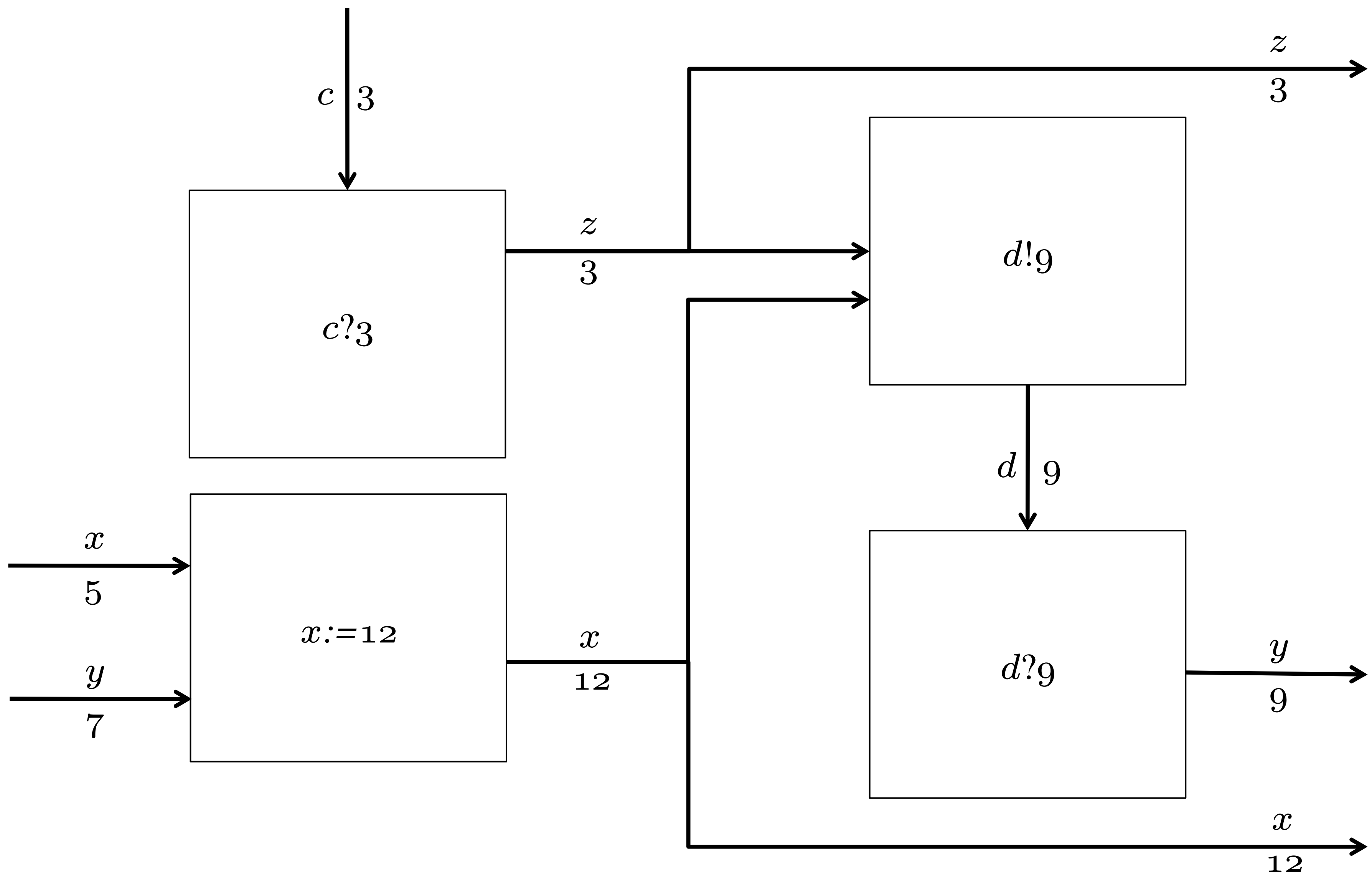
c!x | (b?y . P(y))

CSP Communication



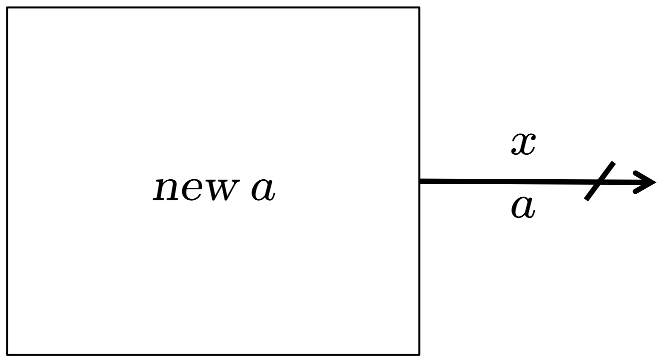
b!x | (b?y . P(y))

Combined Example



(x = x+y) ; (c?z . (d!(x-z) | d?y))

Allocation



Disposal