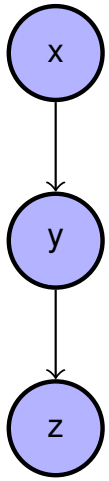
Succinct Representation of Concurrent Trace Sets

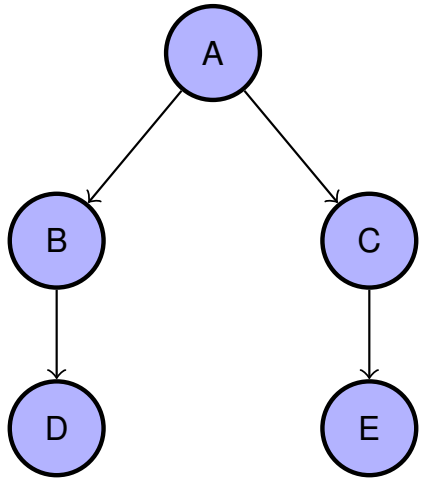
Roopsha Samanta

Joint work with Ashutosh Gupta, Tom Henzinger, Arjun Radhakrishna and Thorsten Tarrach

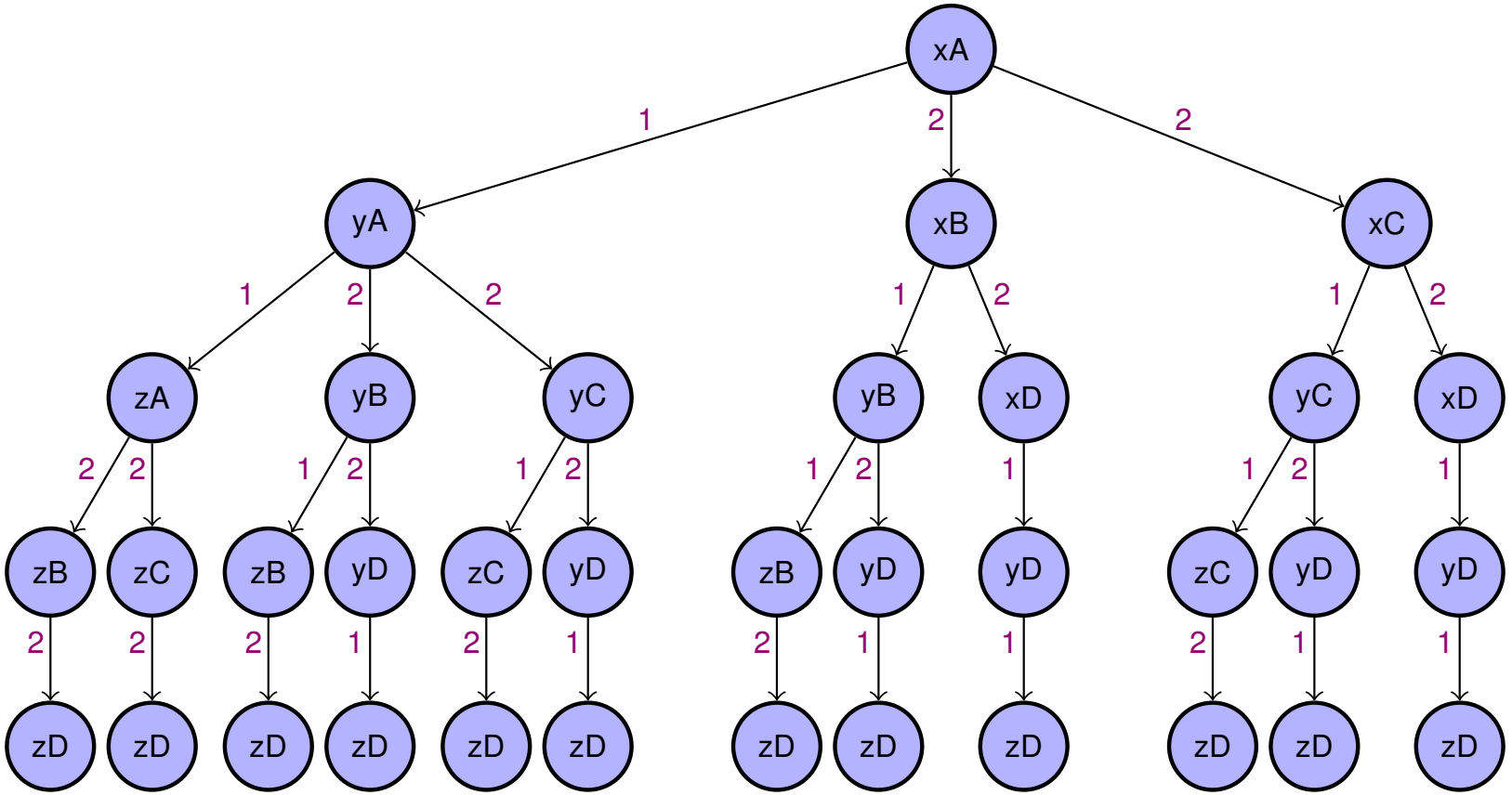
2 December, 2014

Concurrent trace neighbourhoods

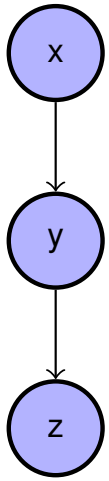


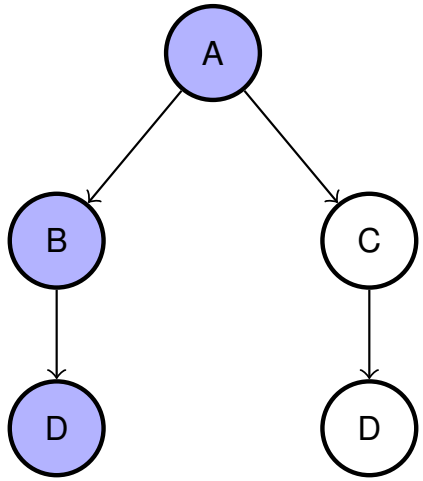


Concurrent trace neighbourhoods

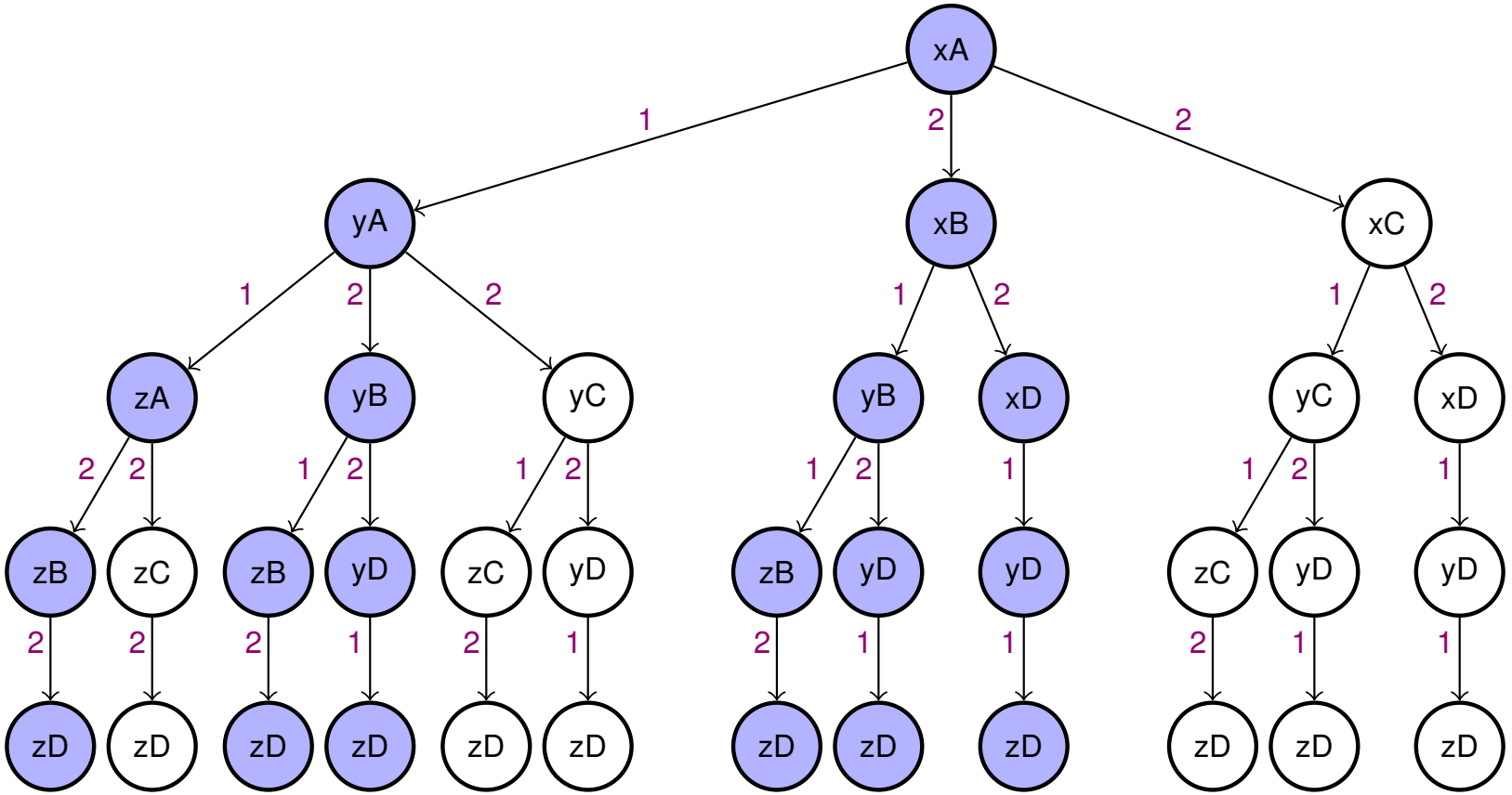


Concurrent trace neighbourhoods

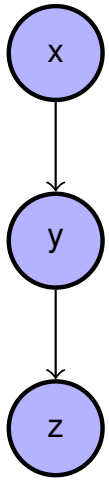


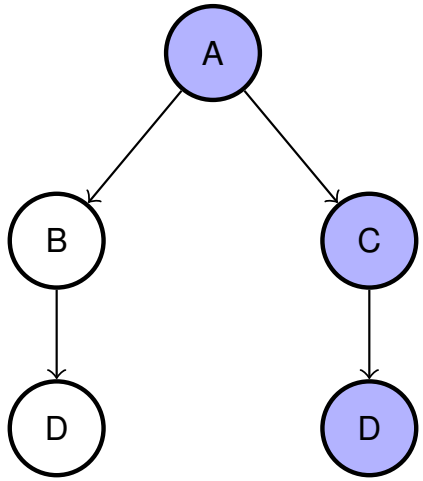


Concurrent trace neighbourhoods

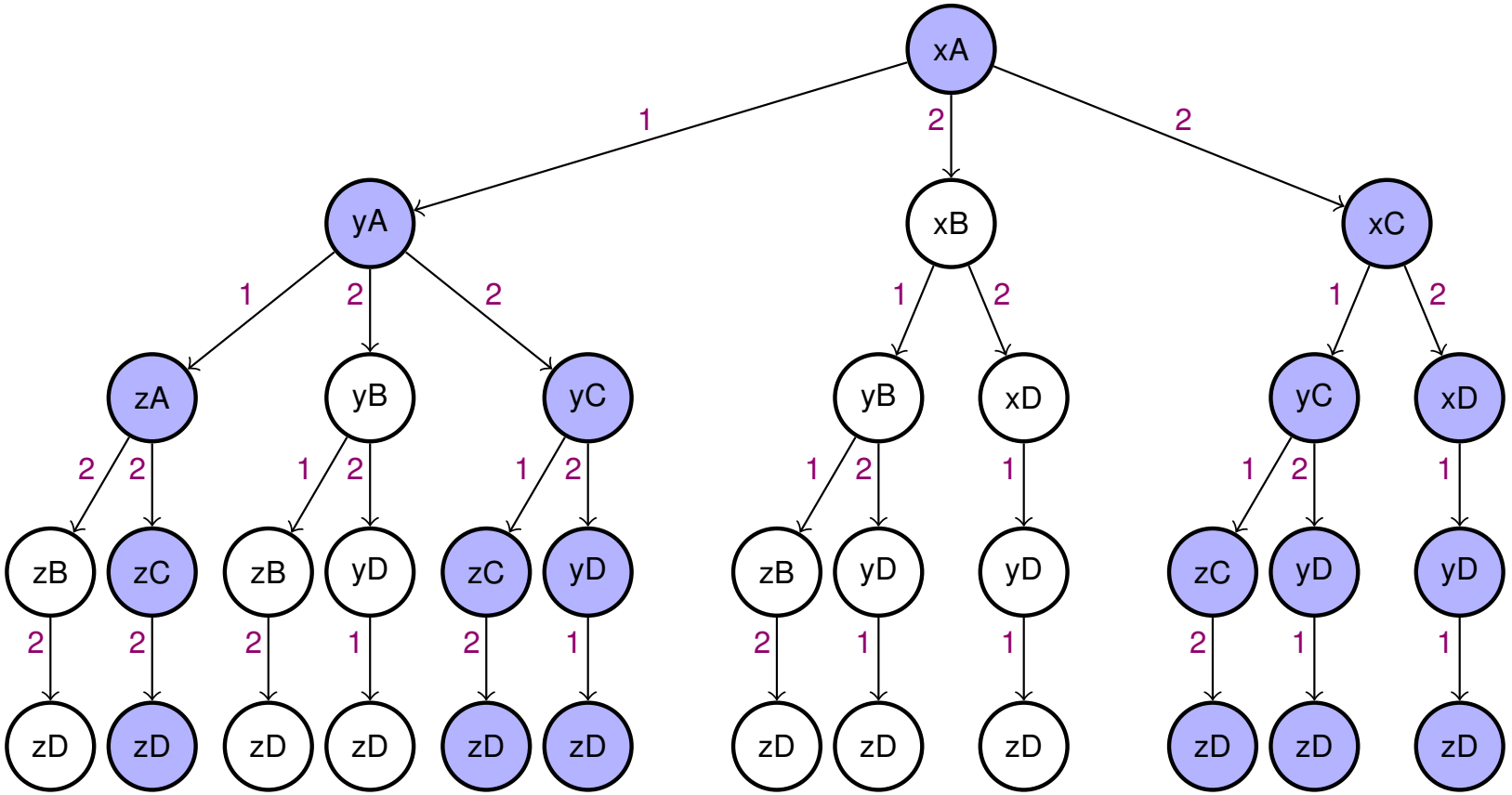


Concurrent trace neighbourhoods

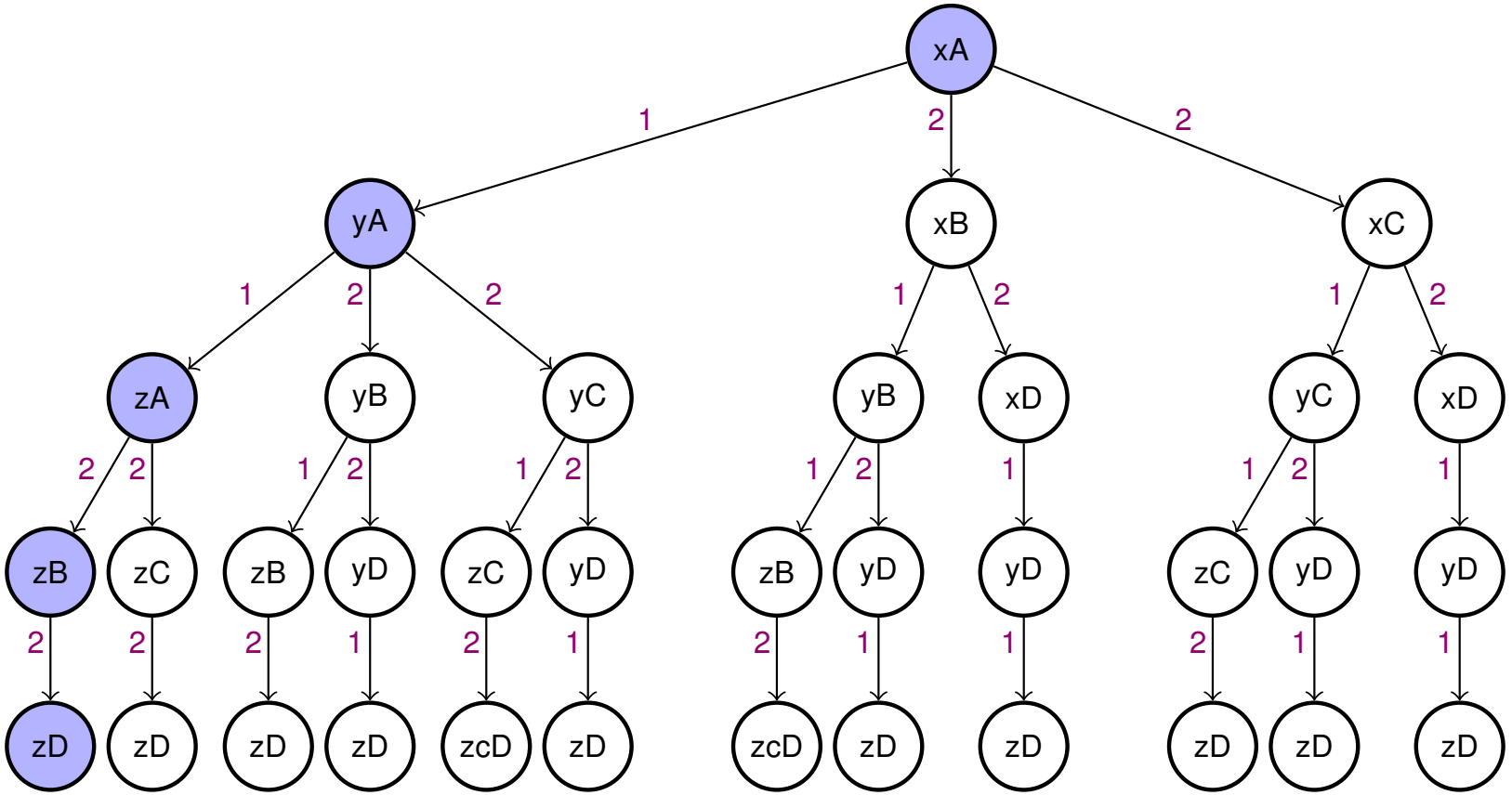




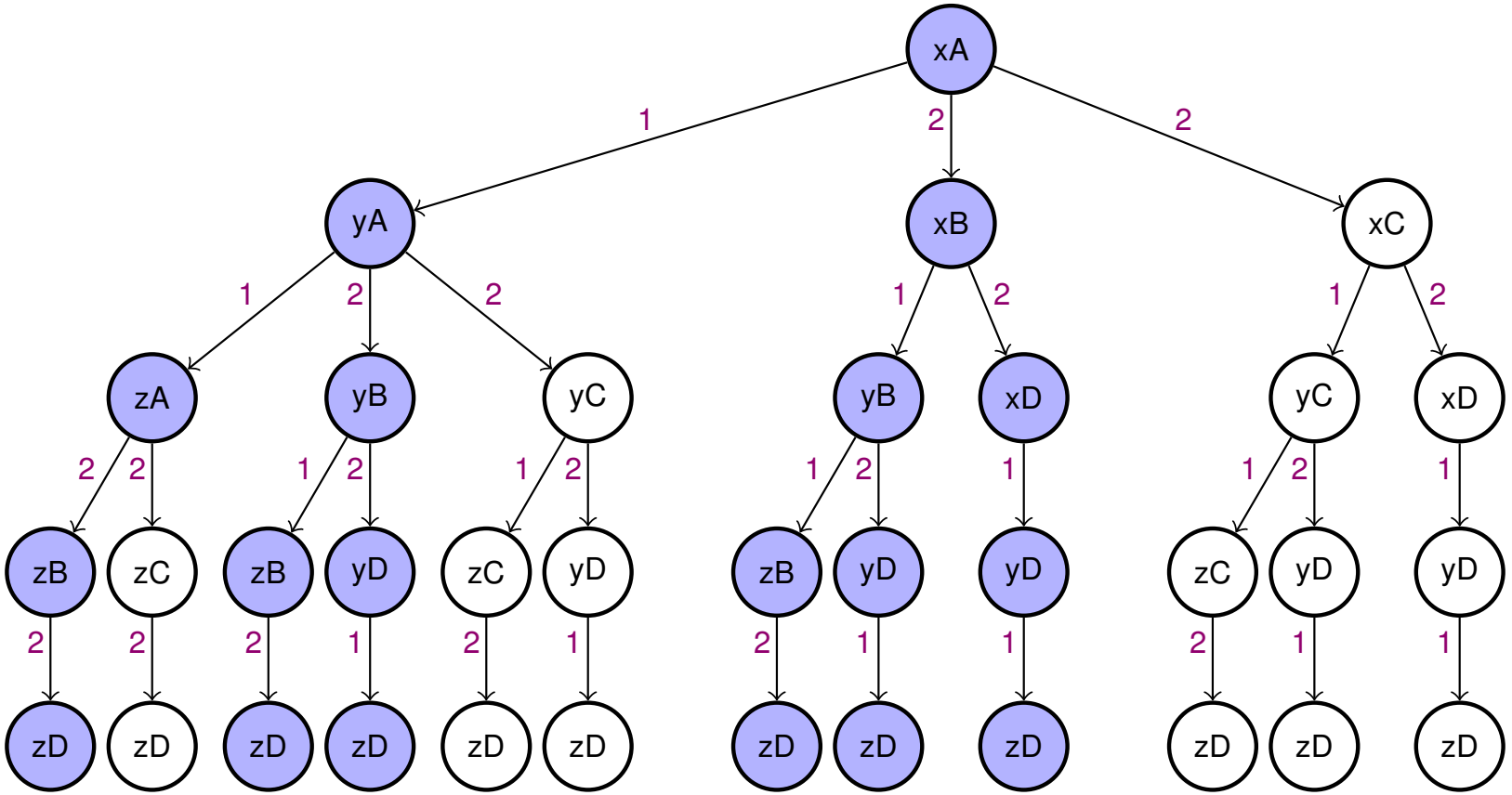
Concurrent trace neighbourhoods



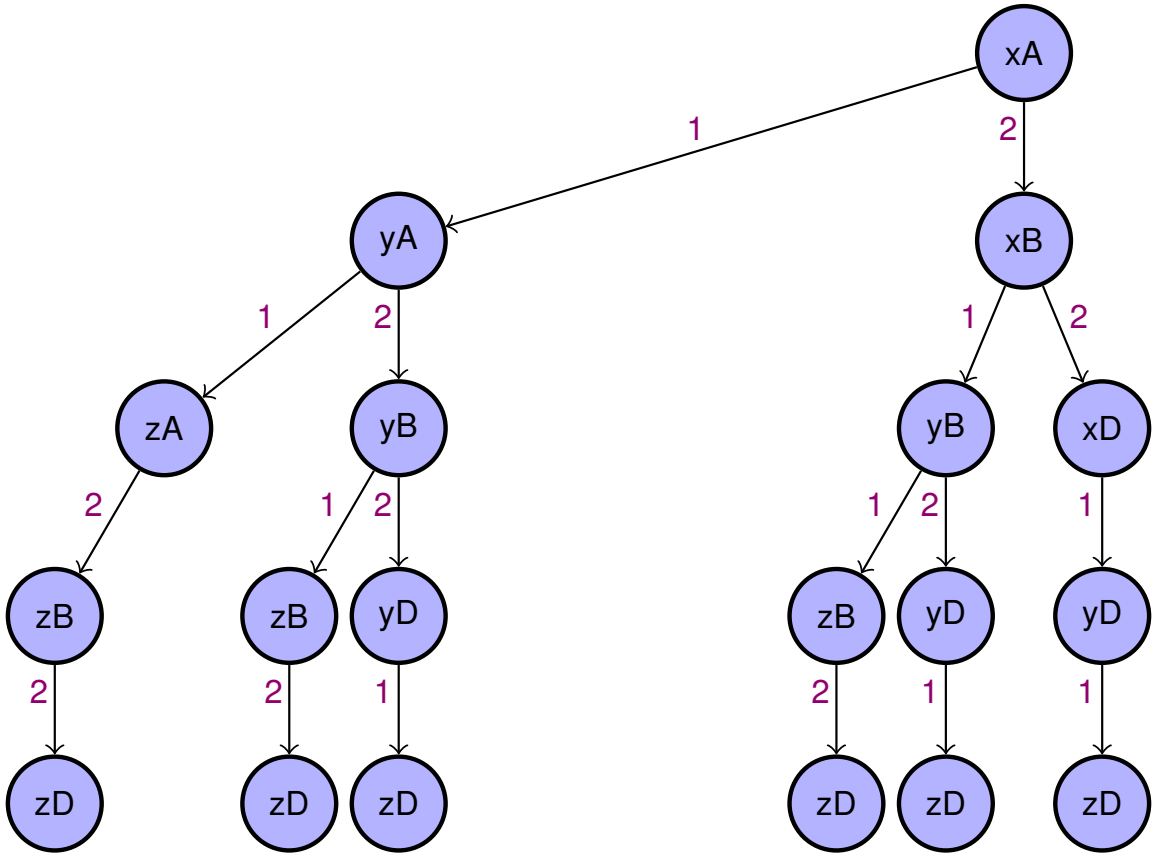
Concurrent trace neighbourhoods



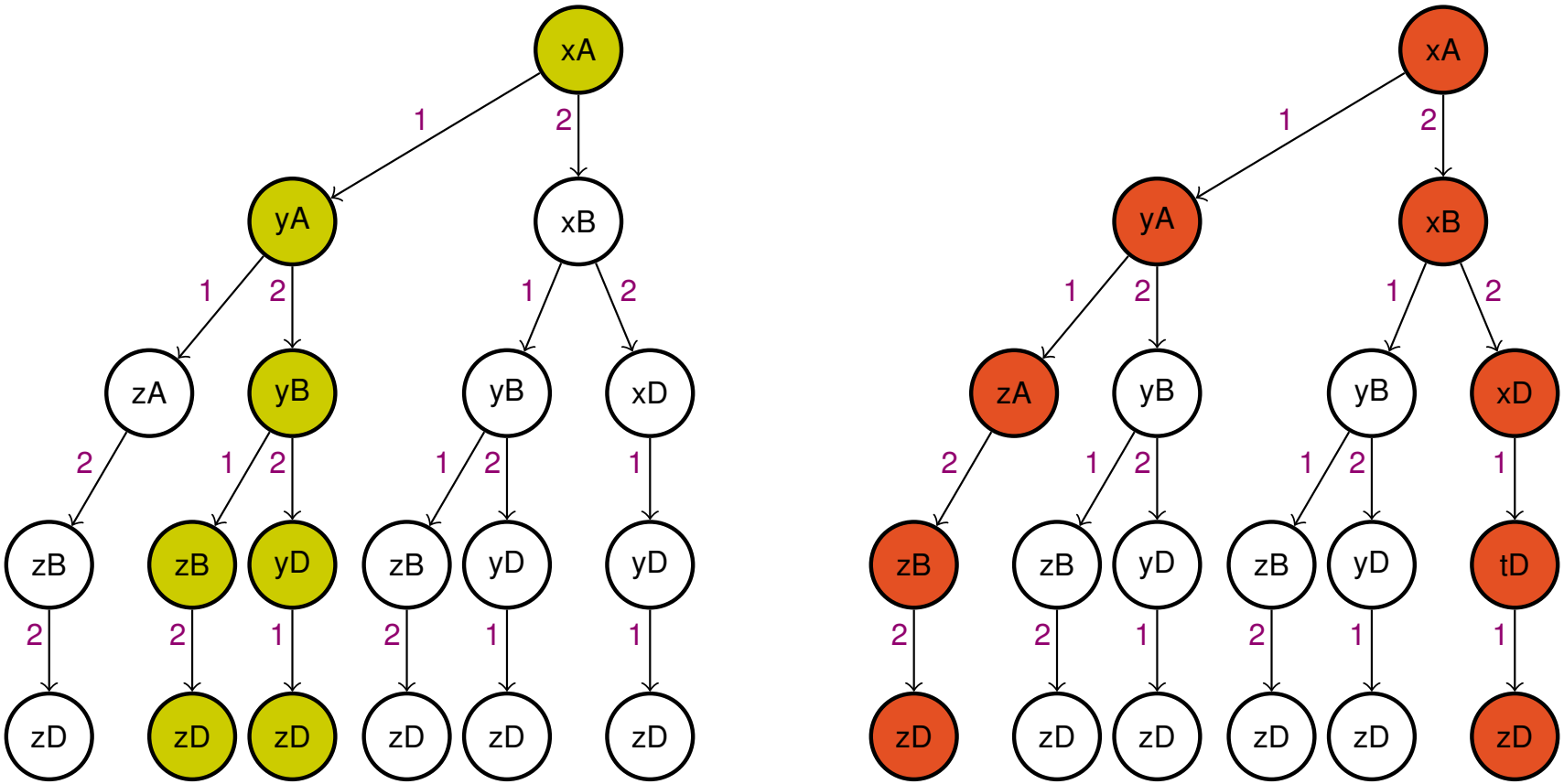
Concurrent trace neighbourhoods



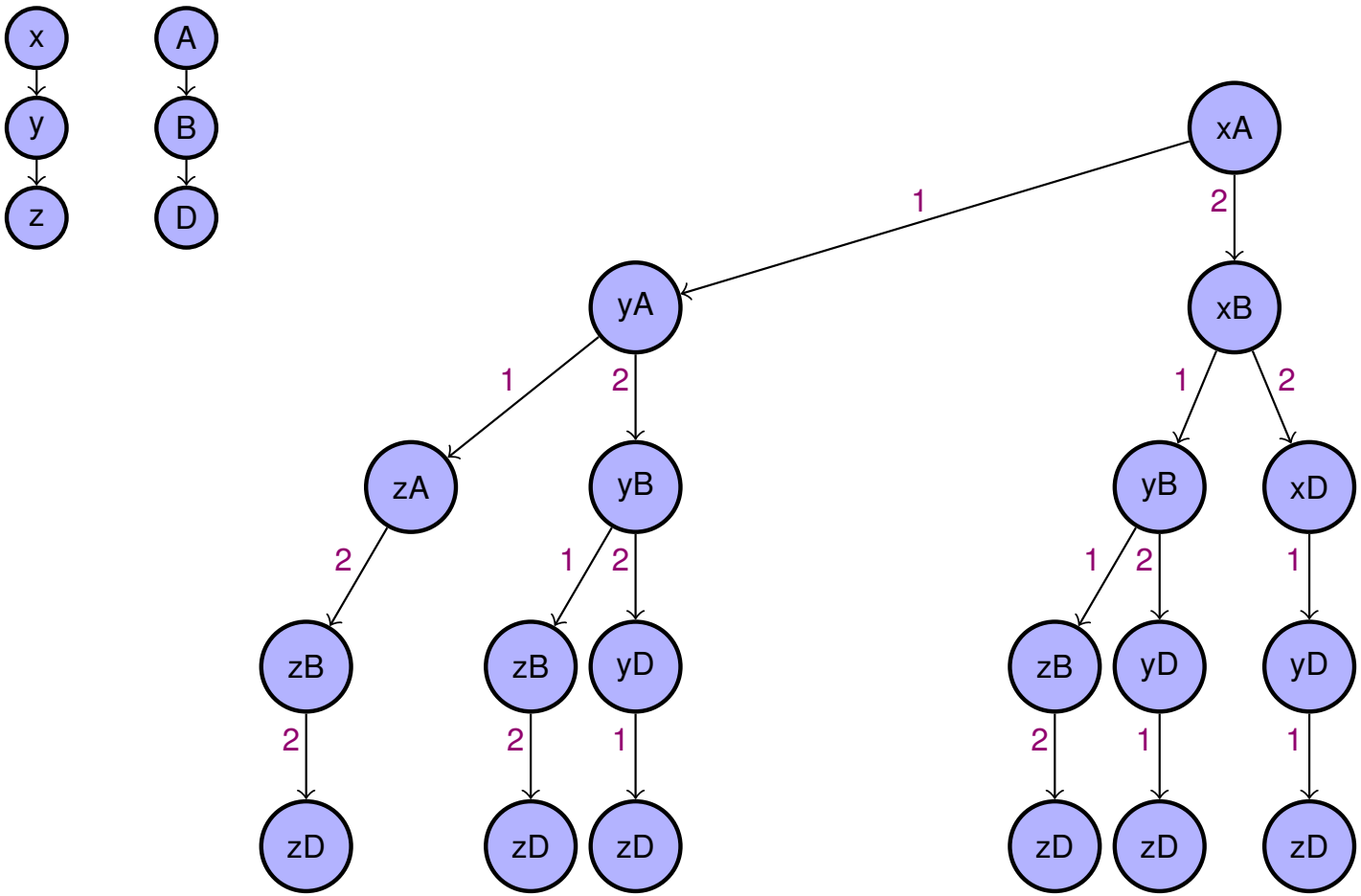
Concurrent trace neighbourhoods



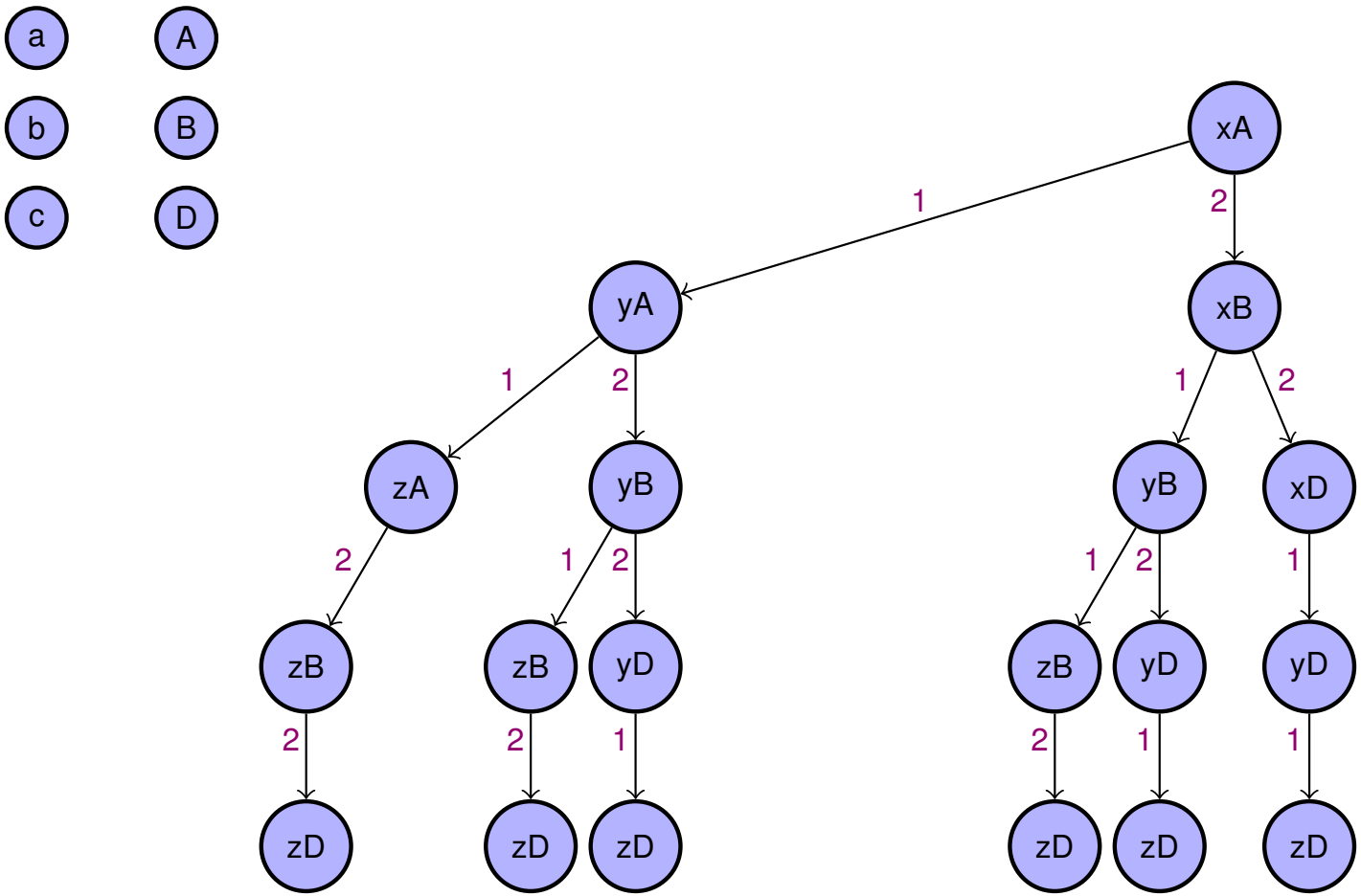
Concurrent trace neighbourhoods



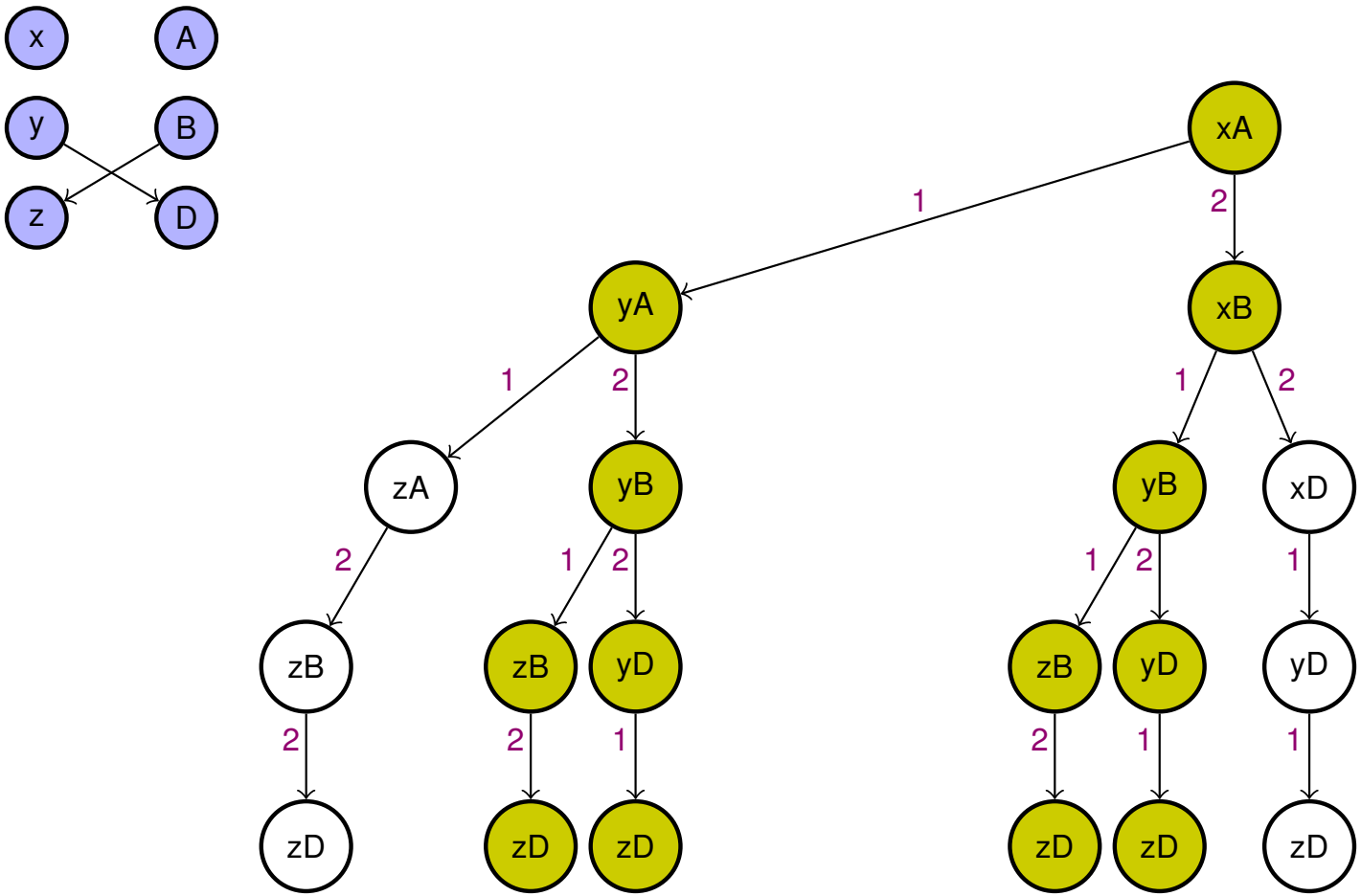
Representation of trace neighbourhoods



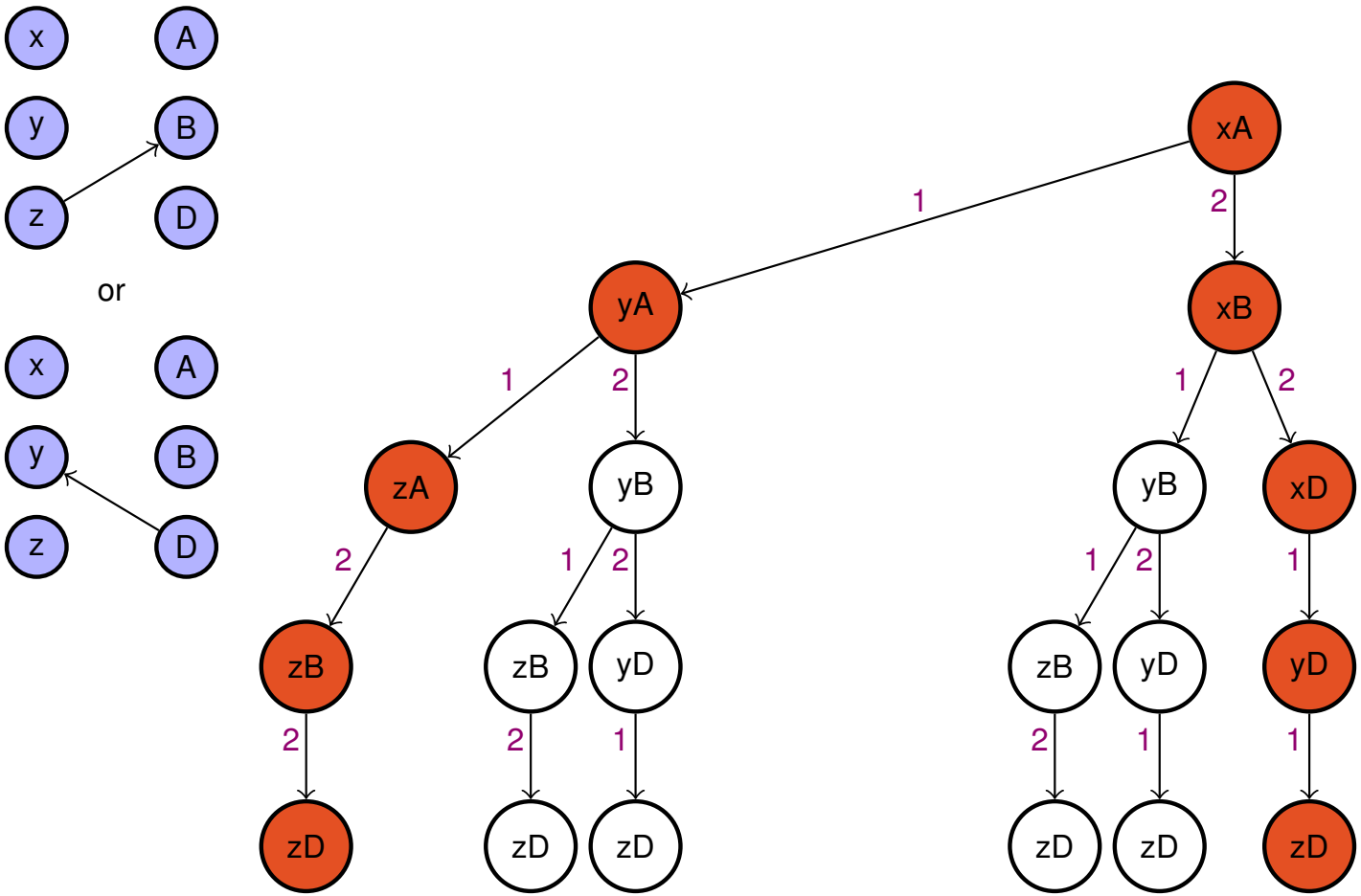
Representation of trace neighbourhoods



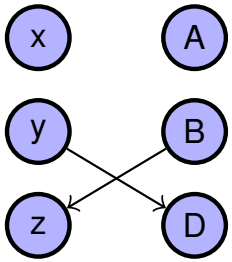
Representation of trace neighbourhoods



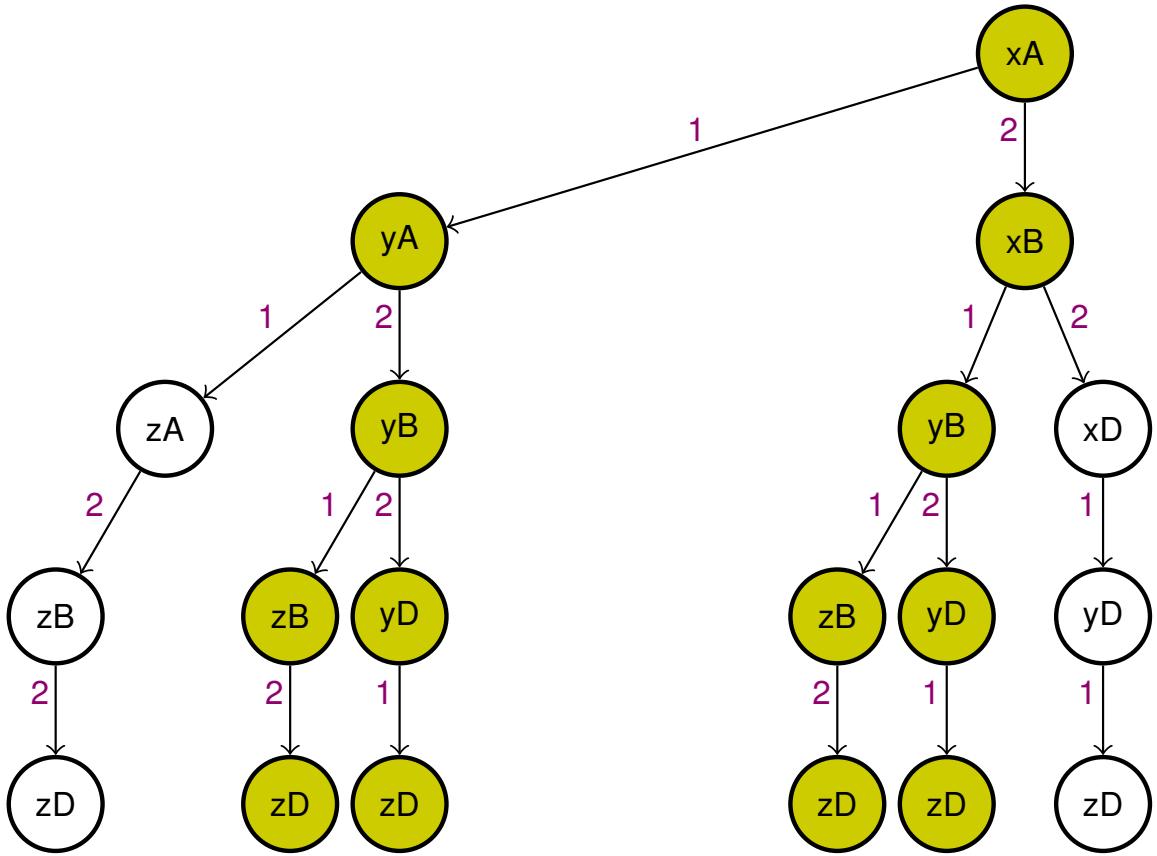
Representation of trace neighbourhoods



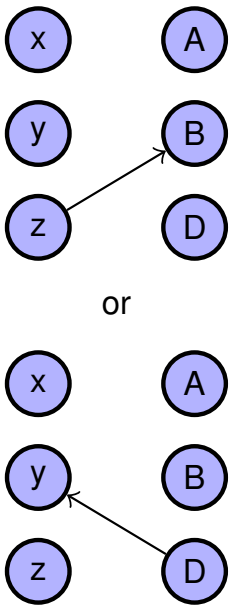
Representation of trace neighbourhoods



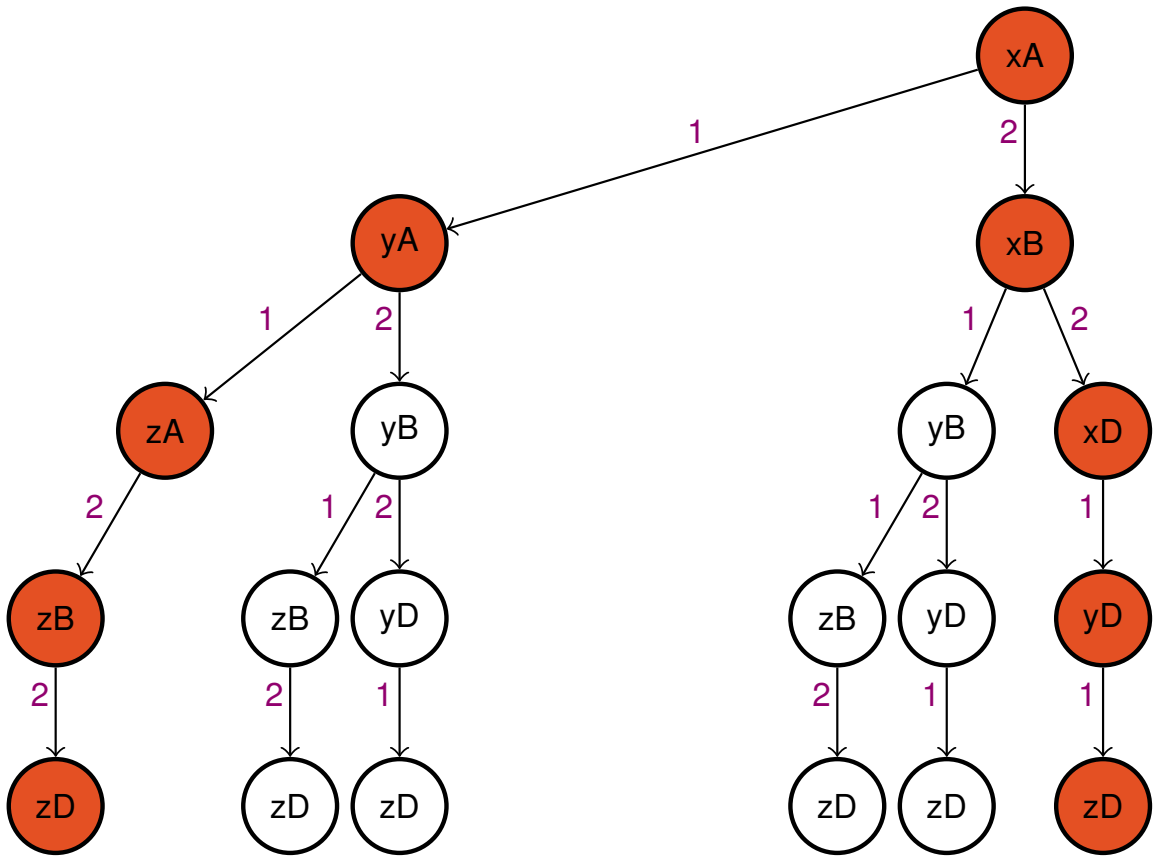
$hb(B, z) \wedge hb(y, D)$



Representation of trace neighbourhoods



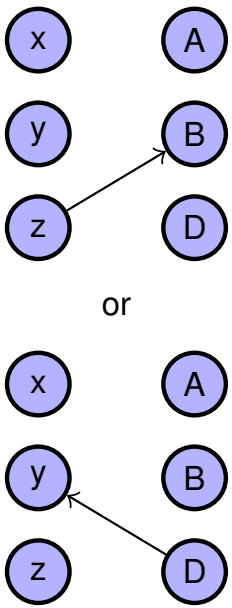
$hb(z, B) \vee hb(D, y)$



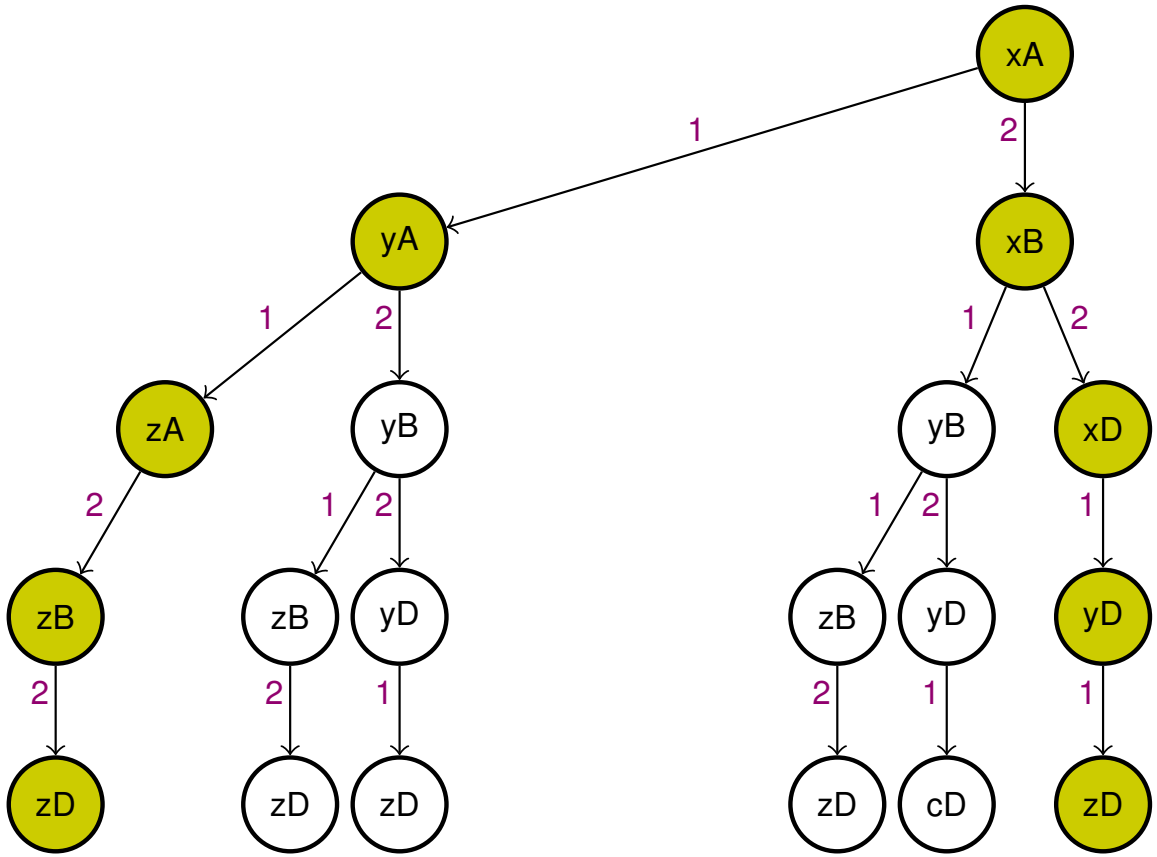
Representation of trace neighbourhoods

- HB-formulas - novel representation for concurrent trace sets
  - Can express arbitrary trace sets
  - Efficient computation of $\cup\{\text{trace sets}\}$
  - Succinct representations of good/bad trace neighbourhoods
  - Intuitively appealing
  - Can drive diverse concurrency applications . . .

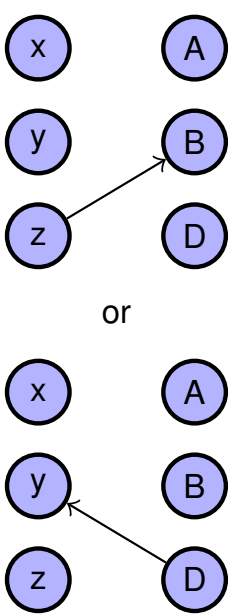
Application: synchronization synthesis



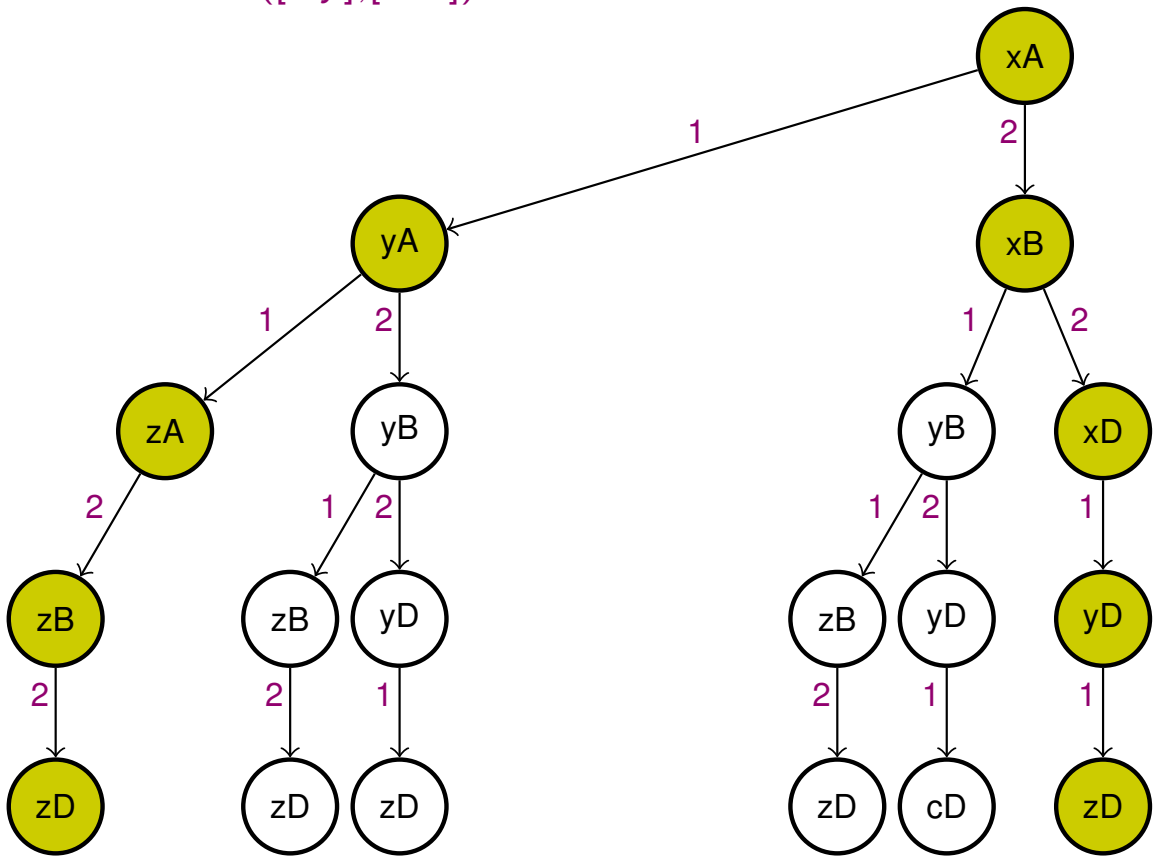
$hb(z, B) \vee hb(D, y)$



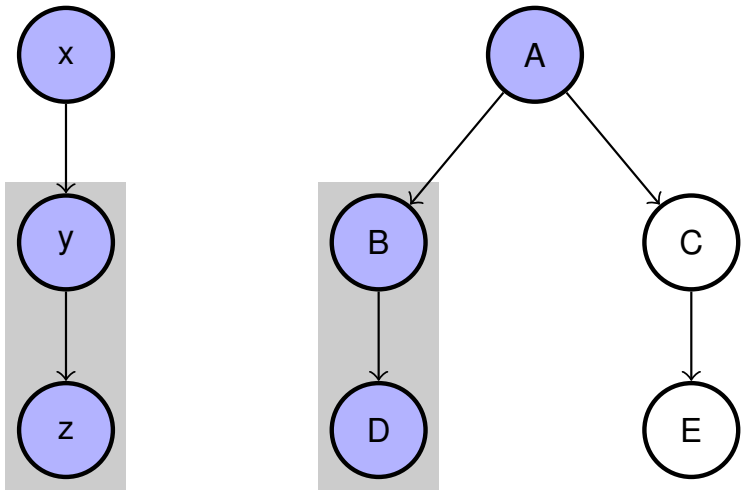
Application: synchronization synthesis



$$\frac{hb(z,B) \vee hb(D,y)}{\texttt{Lock}([x{:}y],[B{:}D])}$$



Application: synchronization synthesis

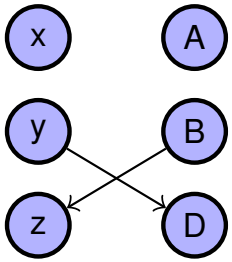


$$\frac{hb(z,B) \vee hb(D,y)}{\texttt{Lock}([x{:}y],[B{:}D])}$$

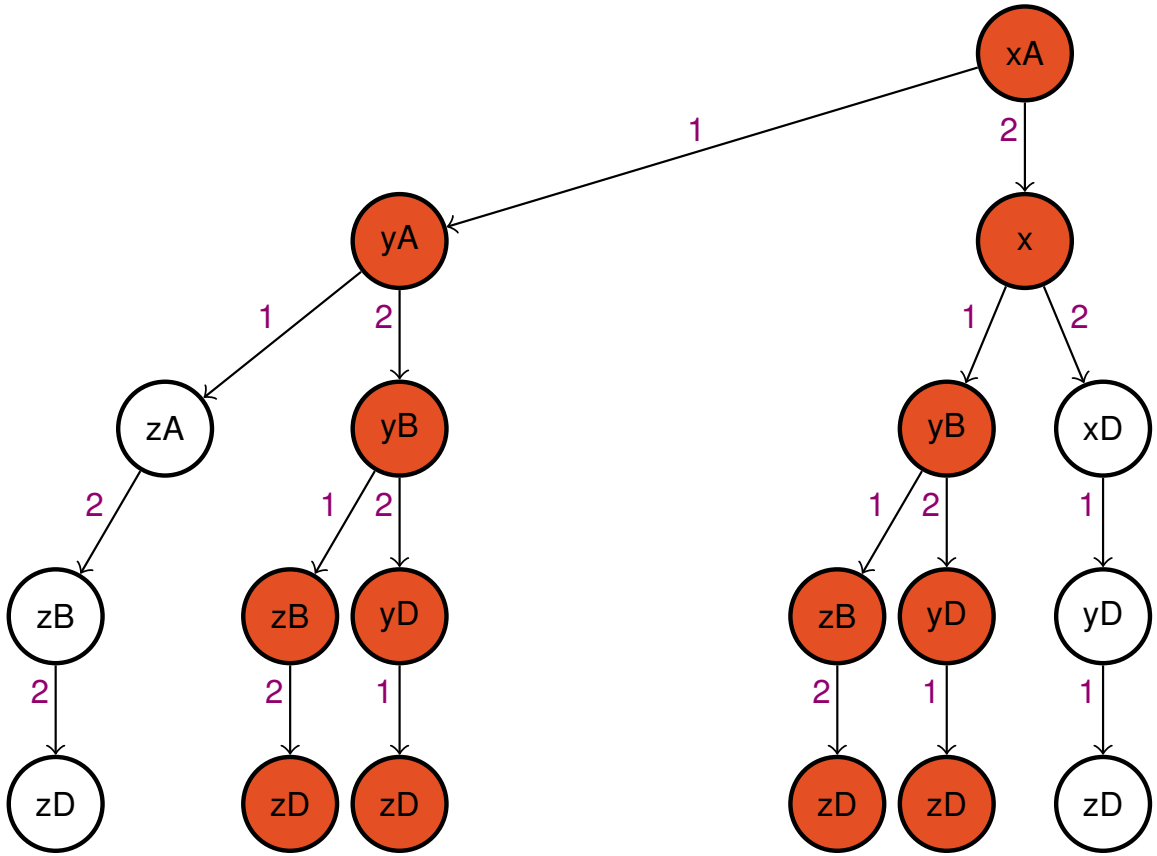
Application: synchronization synthesis

- Automated program completion
- Rewrite rules for synchronization synthesis
  - Identification of HB-formula patterns for sync. primitives
  - Mutex locks, barriers, shared-exclusive locks, wait-notify

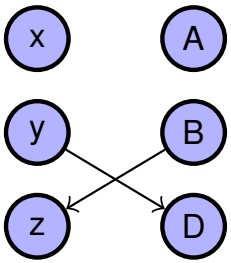
Application: bug summarization



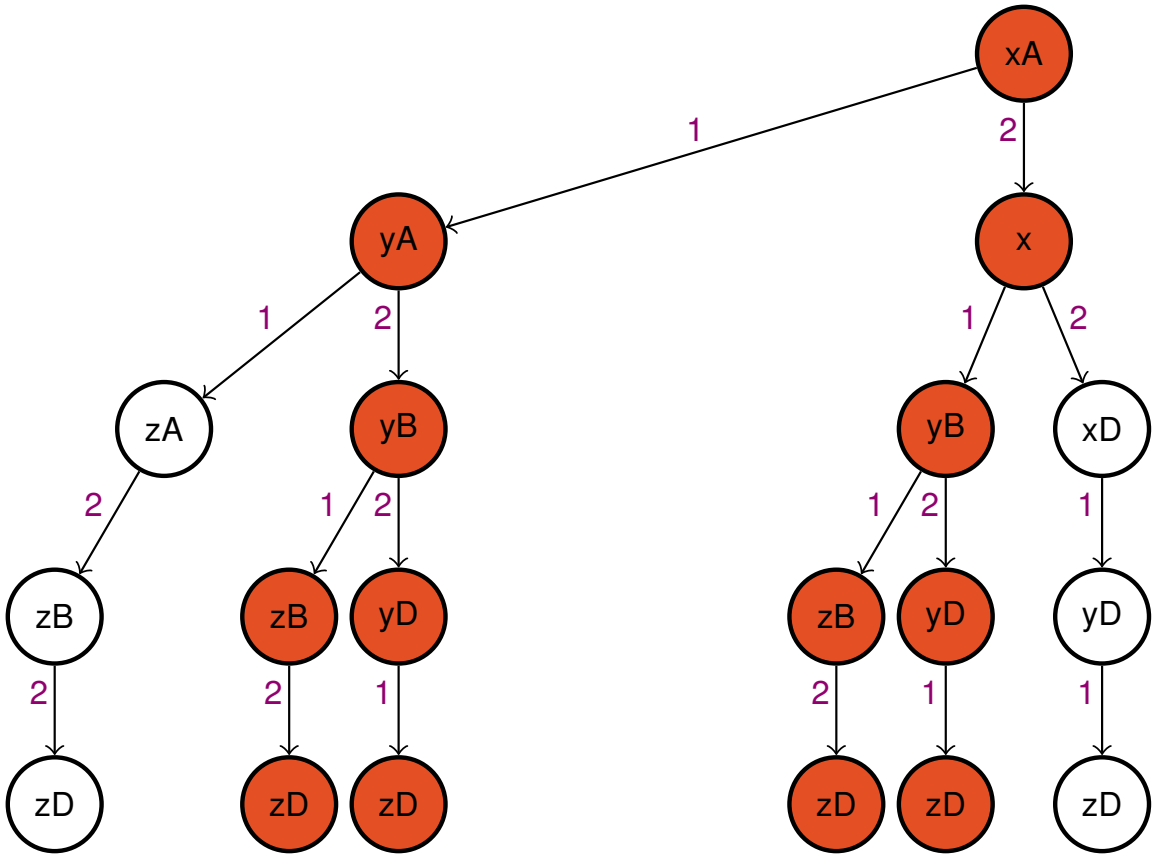
$hb(B, z) \wedge hb(y, D)$



Application: bug summarization



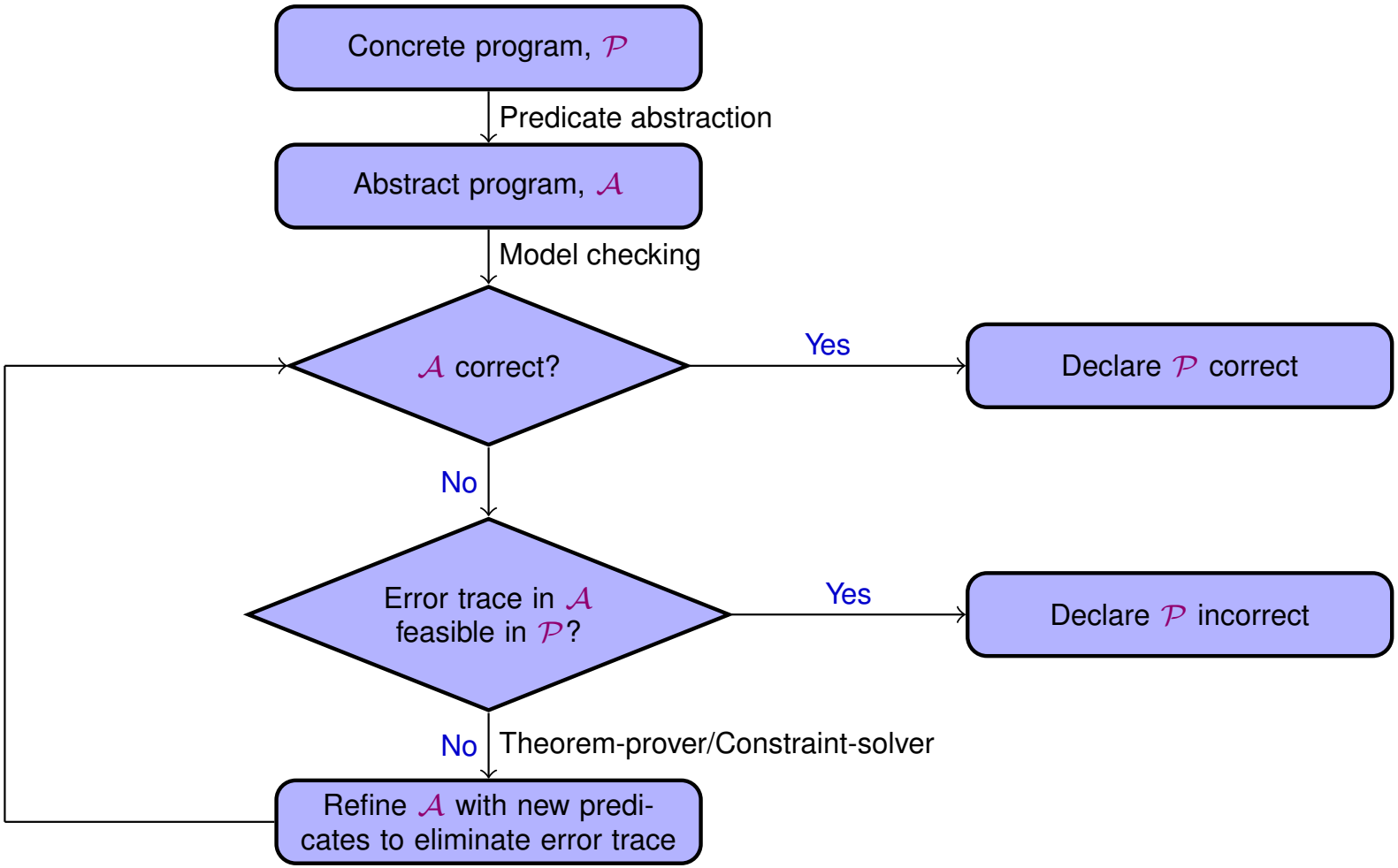
$$\frac{hb(B,z) \wedge hb(y,D)\ access(y,v)\ access(z,v)\ access(B,v)\ access(D,v)}{\texttt{AtomicityViolation}([x{:}y],[B{:}D])}$$



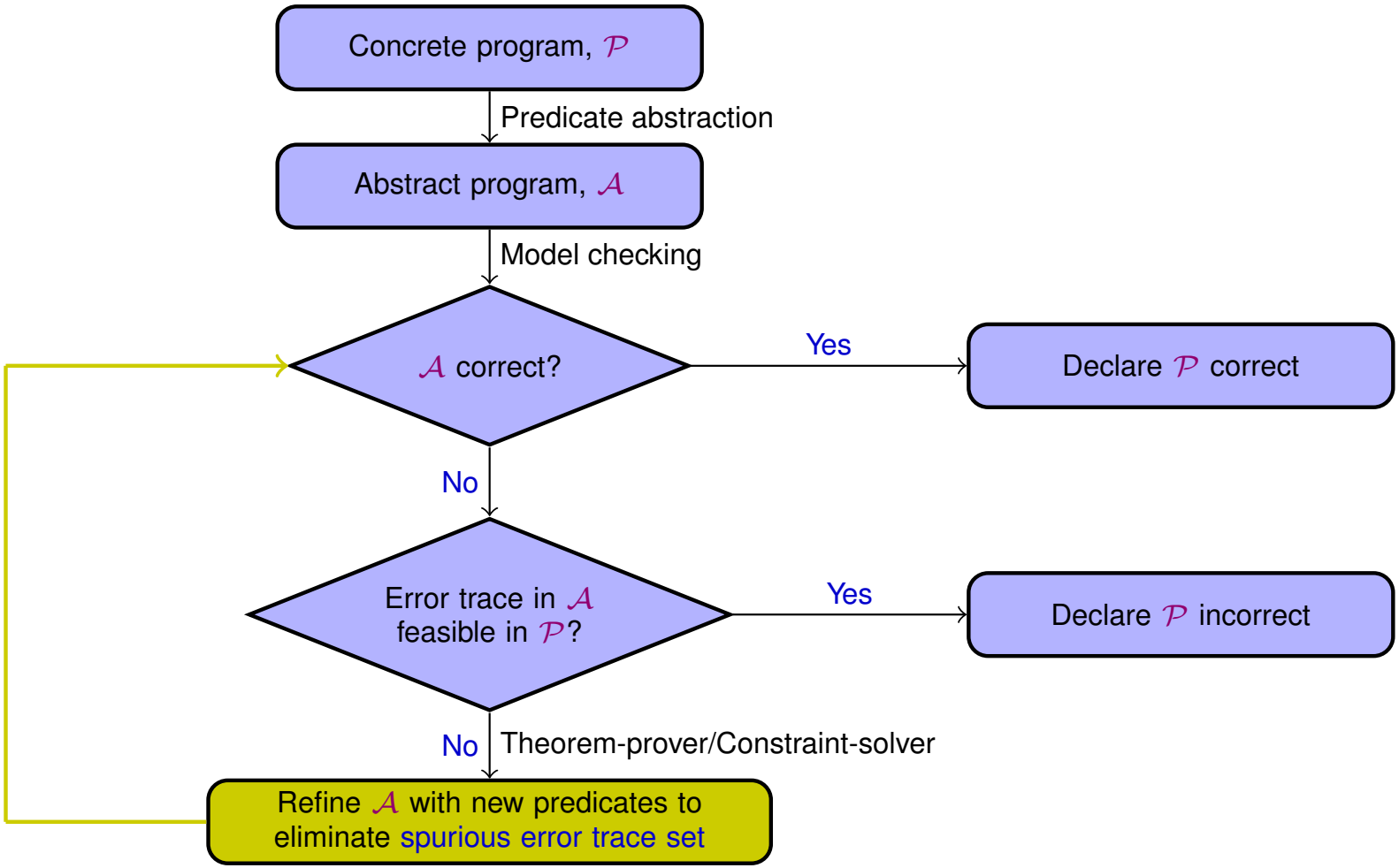
Application: bug summarization

- HB-formula as a counterexample summary
- Inference rules for more precise bug summaries
  - Identification of HB-formula and data access patterns for bugs
  - Data races, atomicity violations, two stage access bugs, define-use order violations

Application: CEGAR acceleration



Application: CEGAR acceleration



Application: CEGAR acceleration

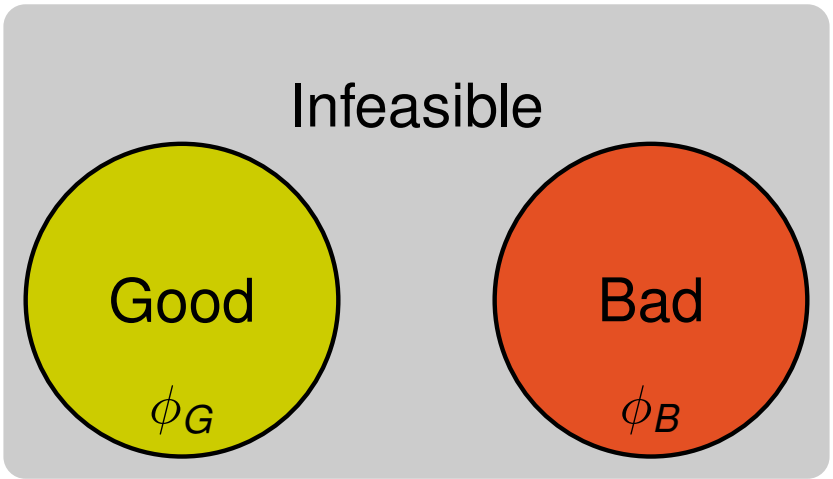
- Representation of abstract, spurious, bad trace sets using HB-formulas
- Simultaneous learning of refinement predicates from multiple abstract cexs

Goal

Given a trace τ and a specification, generate:

- HB-formula ϕ_B representing **nhood**(τ), and,
- HB-formula ϕ_G representing **nhood**(τ).

First attempt



First attempt

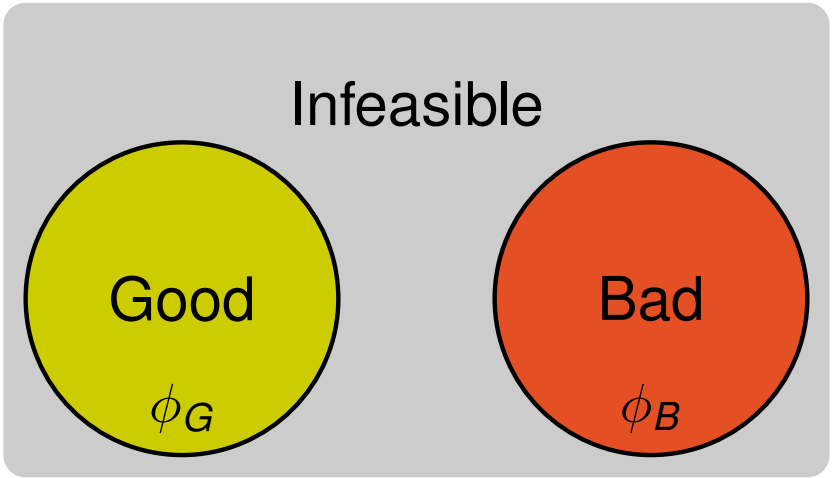
- Formulate Φ [WKGG09]:
  - Quantifier-free first-order formula over vars & *hb* constraints
  - Execution $\pi \in \mathbf{nhood}(\tau)$ iff $\pi \approx$ model of Φ
- To compute ϕ_B:
  - Initially, $\phi_B = \texttt{false}$
  - In each step:
    - Obtain a model π of Φ not subsumed by current ϕ_B
    - Generalize π into an HB-formula φ
    - $\phi_B = \phi_B \vee \varphi$
  - Generalization: *partial satisfying assignments*

ϕ_B exactly represents $\mathbf{nhood}(\tau)$

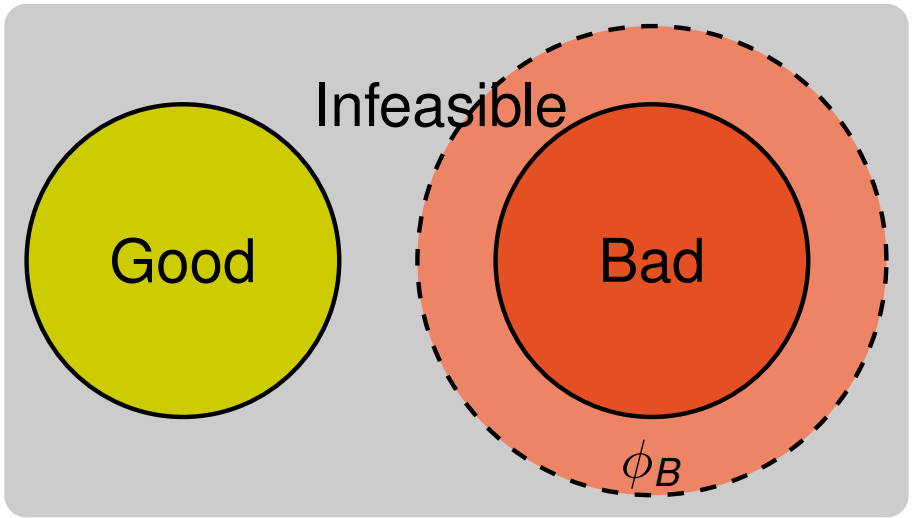
First attempt

- Issues:
  - Inefficient in practice
  - Unclear how to generate ϕ_G

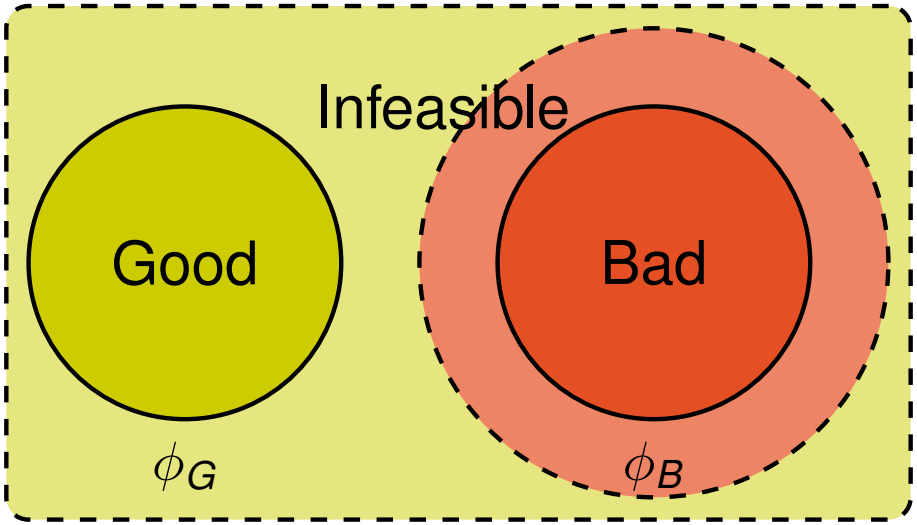
Second attempt



Second attempt



Second attempt



Second attempt

- Formulate Φ [WKGG09]:
  - Quantifier-free first-order formula over vars & *hb* constraints
  - Execution $\pi \in$ **nhood**(τ) iff $\pi \approx$ model of Φ
- To compute ϕ_B:
  - Initially, $\phi_B = \texttt{false}$
  - In each step:
    - Obtain a model π of Φ not subsumed by current ϕ_B
    - Generalize π into an HB-formula φ
    - $\phi_B = \phi_B \vee \varphi$
  - Generalization: *data-flow analysis + minimal unsat core*

ϕ_B soundly overapproximates **nhood**(τ)

Second attempt

- Formulate Φ [WKGG09]:
  - Quantifier-free first-order formula over vars & *hb* constraints
  - Execution $\pi \in$ **nhood**(τ) iff $\pi \approx$ model of Φ
- To compute ϕ_B:
  - Initially, $\phi_B = \texttt{false}$
  - In each step:
    - Obtain a model π of Φ not subsumed by current ϕ_B
    - Generalize π into an HB-formula φ
    - $\phi_B = \phi_B \vee \varphi$
  - Generalization: data-flow analysis + minimal unsat core
- To compute ϕ_G: $\neg\phi_B$

ϕ_G soundly overapproximates the good neighbourhood of τ

Experiments using TARA for ϕ_B generation

Name	#P/#I	#π/#Disj.	Iterations		Total time		Size of ϕ_B	
			Alg.1	Alg.2	Alg.1	Alg.2	Alg.1	Alg.2
reorder_2	2/3	2/2.0	1	1	18ms	28ms	1/2.0	1/2.0
define_use	2/4	2/2.0	1	1	15ms	22ms	1/2.0	1/1.0
em28xx	2/8	4/2.0	1	1	16ms	25ms	1/2.0	1/1.0
locks	3/8	10/1.6	12	2	27ms	37ms	12/5.5	2/4.0
2stage	2/8	5/1.4	8	1	26ms	32ms	8/3.8	1/2.0
drbd_receiver	2/9	5/1.6	40	1	42ms	28ms	40/3.9	1/1.0
md	3/11	4/1.8	40	1	76ms	33ms	40/6.1	1/1.0
lazy01	3/12	6/3.7	2	2	31ms	57ms	2/3.0	2/2.0
locks_hb	4/13	10/2.2	>29.0k	7	TO	119ms	TO	6/3.0
lc_rc	4/14	8/2.0	4.6k	1	21.4s	37ms	4.6k/16.7	1/1.0
barrier_locks	3/18	17/2.6	10.6k	6	1.4min	521ms	10.6k/10.0	4/1.5
stateful01	3/19	10/3.4	2.3k	2	10.5s	84ms	2.3k/9.4	2/1.0
read_write_lock	4/22	16/3.4	9.2k	4	1.6min	319ms	9.2k/16.1	4/3.0
loop	2/38	14/2.7	2	1	38ms	72ms	2/3.0	1/2.0
fib_bench	3/39	24/3.6	>20.5k	2	TO	2.3s	TO	2/10.0
i2c_hid	2/42	26/4.5	>23.4k	3	TO	615ms	TO	3/1.3
rtl8169-1	7/71	22/2.7	>20.4k	1	TO	111ms	TO	1/2.0
rtl8169-2	7/116	41/2.3	>7.3k	1	TO	463ms	TO	1/1.0
rtl8169-5	7/134	48/3.1	>5.5k	1	TO	1.5s	TO	1/1.0
rtl8169-4	7/142	48/3.0	>8.4k	9	TO	3.8s	TO	2/1.0
rtl8169-6	7/144	52/2.9	>8.1k	1	TO	887ms	TO	1/1.0
usb_serial-1	7/151	87/3.7	>5.5k	1	TO	1.9s	TO	1/1.0
usb_serial-2	7/163	93/3.6	>4.4k	3	TO	4.4s	TO	1/1.0
rtl8169-3	8/174	61/3.6	>4.2k	2	TO	2.7s	TO	1/1.0
usb_serial-3	7/178	100/3.7	>4.3k	1	TO	2.1s	TO	1/1.0

Goal

Given a trace τ and a specification,
synthesize synchronization to eliminate **nhood**(τ).

Basic idea

- Use ϕ_G (in CNF)
- Identify HB-formula patterns for various synchronization primitives
- Formulate rewrite rules
- Repeatedly rewrite patterns into synchronization primitives
- Obtain CNF formula over synchronization primitives
- Pick a set $\mathcal{S}$ of synchronization pritmitives, one from each conjunct

Examples

$$\frac{\mathit{hb}(\mathtt{T}_1[\ell_1], \mathtt{T}_2[\ell_2]) \vee \psi}{\mathtt{WaitNotify}(\mathtt{T}_2[\ell_2], \mathtt{T}_1[\ell_1]) \vee \psi} \text{ ADD.WAITNOTIFY}$$

Examples

$$\frac{\mathit{hb}(\mathtt{T}_1[\ell'_1], \mathtt{T}_2[\ell_2]) \vee \mathit{hb}(\mathtt{T}_2[\ell'_2], \mathtt{T}_1[\ell_1]) \vee \psi \quad \ell_1 \leq \ell'_1 \quad \ell_2 \leq \ell'_2}{\mathtt{Lk}\left(\mathtt{T}_1[\ell_1 : \ell'_1], \mathtt{T}_2[\ell_2 : \ell'_2]\right) \vee \psi} \textsc{Add.Lock}$$

Examples

$$\frac{\left(\mathit{hb}(\mathtt{T}_1[\ell_1 - 1], \mathtt{T}_2[\ell_2]) \vee \psi\right) \wedge \left(\mathit{hb}(\mathtt{T}_2[\ell_2 - 1], \mathtt{T}_1[\ell_1]) \vee \psi\right)}{\mathtt{Barrier}\left(\mathtt{T}_1[\ell_1], \mathtt{T}_2[\ell_2]\right) \vee \psi} \quad \textsc{Add.Barrier}$$

Examples

Additional rewrite rules for:

- Shared exclusive locks
- Multithreaded locks
- Multithreaded barriers
- Merging locks (to avoid deadlocks)

Soundness of rewrite rules

Given a trace τ of a concurrent program $\mathcal{P}$,
let $\mathcal{P}^{\mathcal{S}}$ be obtained by inserting synchronization primitives from $\mathcal{S}$.
Let $\pi \in \textbf{nhood}(\tau)$ be a deadlock-free execution of $\mathcal{P}^{\mathcal{S}}$.
Then, π is not bad.

Experiments

Name	#L	#B	#WN	Name	#L	#B	#WN
reorder_2	1	0	0	loop	1	0	0
define_use	0	0	1	fib_bench	1	0	0
em28xx	0	0	1	i2c_hid	1	0	2
locks	1	0	0	rtl8169-1	0	0	1
2stage	0	0	1	rtl8169-2	0	0	1
drbd_receiver	0	0	1	rtl8169-5	0	0	1
md	0	0	1	rtl8169-4	0	0	2
lazy01	0	0	2	rtl8169-6	0	0	1
locks_hb	1	0	2	usb_serial-1	0	0	1
lc_rc	0	0	1	usb_serial-2	0	0	1
barrier_locks	1	1	0	rtl8169-3	0	0	1
stateful01	0	0	2	usb_serial-3	0	0	1
read_write_lock	4	0	0				

Summary

- A method and a tool TARA for succinct representations of sound overapproximations of **nhood**(τ) and **nhood**(τ)
- Three successful case studies using TARA
  - Synchronization synthesis
  - Bug summarization
  - CEGAR acceleration
- Other applications?

Thank you.