Efficient Reasoning about Complex Data Types

Viorica Sofronie-Stokkermans

Max-Planck-Institut für Informatik

Saarbrücken

(joint work with Carsten Ihlemann and Swen Jacobs)

Deduktionstreffen 2008, Saarbrücken, March 17, 2008

Introduction

Motivation

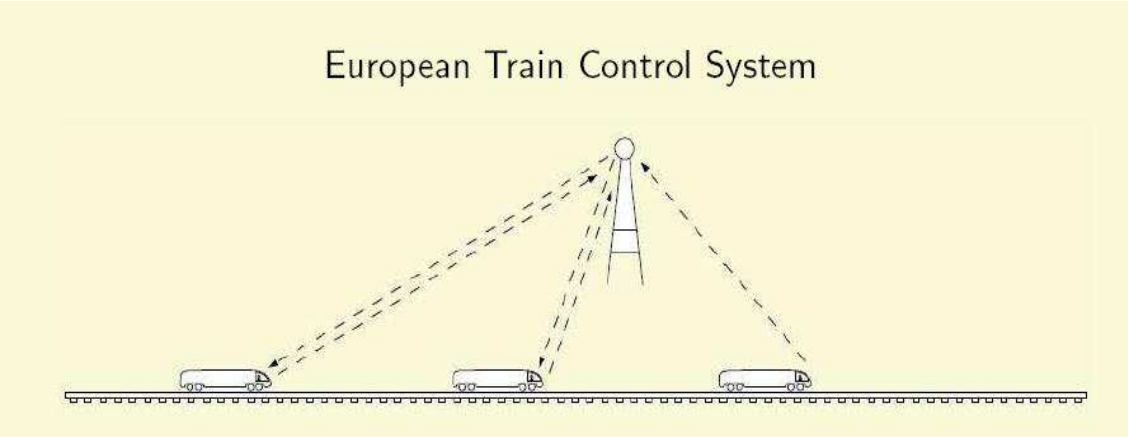
– program verification (use data structures such as lists, arrays, pointers)

– verification of parametric systems

Examples:

– Systems with an unspecified, parametric number of components

(trains on a line track; processes which need to be enabled/disabled)



process nr.	1	2	3	4	5	6	7	···
enabled (0/1)	1	0	0	0	0	0	0	···
priority	0	7	3	1	9	4	2	···

process nr.	1	2	3	4	5	6	7	···
enabled (0/1)	0	0	0	0	1	0	0	···
priority	1	8	4	2	0	5	3	···

Introduction

Motivation

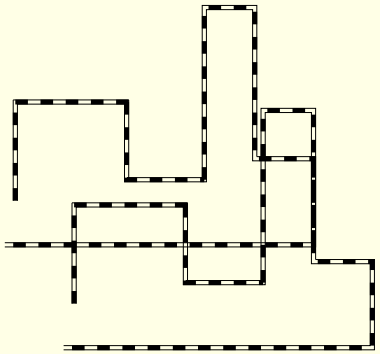
– program verification (use data structures such as lists, arrays, pointers)

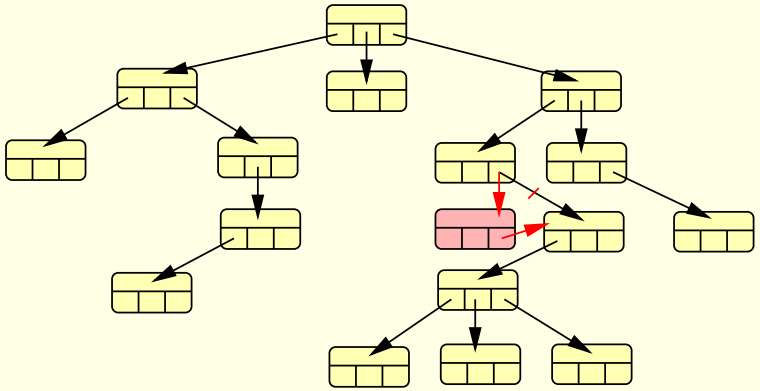
– verification of parametric systems

Examples:

– Topology of a system is parametric and/or changes

('fixed'/'mutable' networks)





Introduction

Motivation

- program verification (use data structures such as lists, arrays, pointers)
- verification of parametric systems

Our goal: Efficient reasoning about complex data types

- Modular reasoning in complex theories: Methodological framework
- Identify decidable and tractable theories resp. problems
- Efficient implementation

Main idea: Restrict search and exploit modularity

Emphasis: efficient, sound & complete decision procedures for certain fragments

Recent results

- Efficient decision procedures for theories of pointer structures and arrays
- Extends the results of [Necula, McPeak, CAV 2005] (pointers)
  [Bradley,Manna,Sipma, VMCAI 2006] (arrays)

Idea [Ihlemann, Jacobs, VS, TACAS 2008]

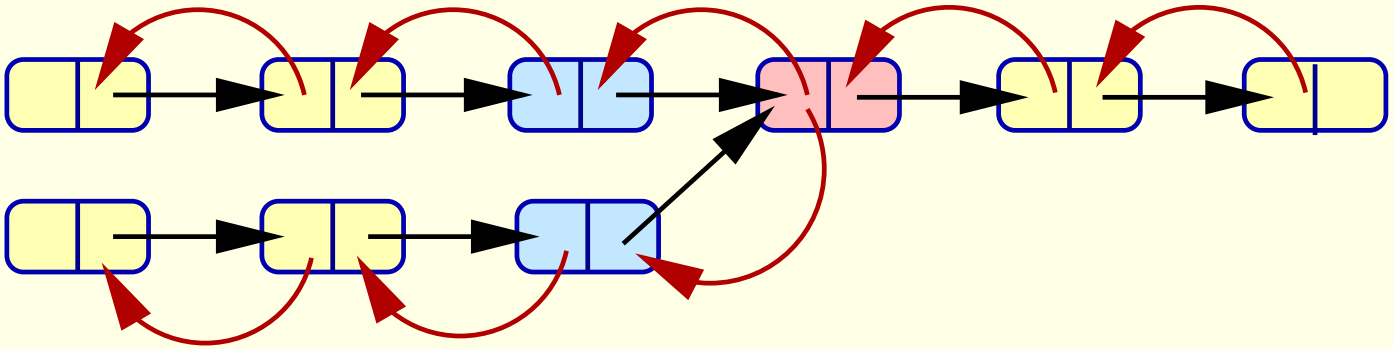
- Identify "local" axiom classes,
  i.e. classes in which search space can be controlled without loss of completeness
- Consider successive extensions with axioms in such classes
- Consider combinations of extensions

Main contribution

- Methodological framework for modular reasoning in complex theories: considerable advance compared with the state of the art
- Efficient implementation $\mapsto$ Carsten's talk
- Applications to the verification of parametric systems $\mapsto$ Swen's talk

Locality: Idea

Reasoning about doubly-linked lists cf. also [Necula, McPeak 2005]



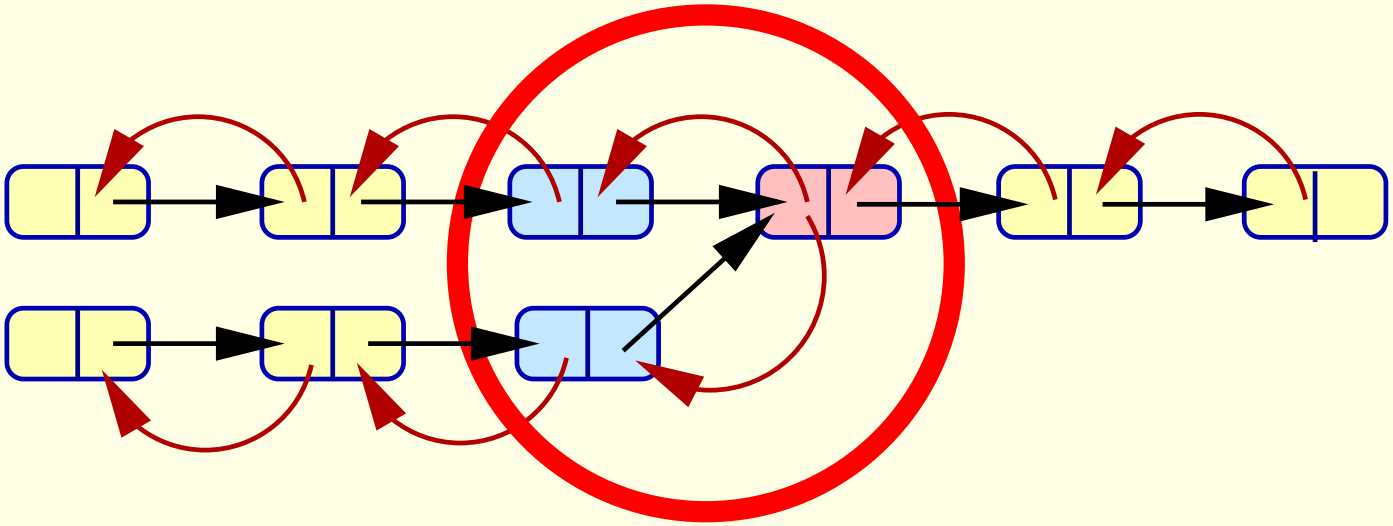
$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{next.prev} = p)$$

$$\forall p\ (p \neq \text{null} \wedge p.\text{prev} \neq \text{null} \ \rightarrow\ p.\text{prev.next} = p)$$

$$\wedge\ \ c{\neq}\text{null} \wedge c.\text{next}{\neq}\text{null} \wedge d{\neq}\text{null} \wedge d.\text{next}{\neq}\text{null} \wedge c.\text{next}{=}d.\text{next} \wedge c \neq d \quad \models \quad \bot$$

Locality: Idea

Reasoning about doubly-linked lists cf. also [Necula, McPeak 2005]



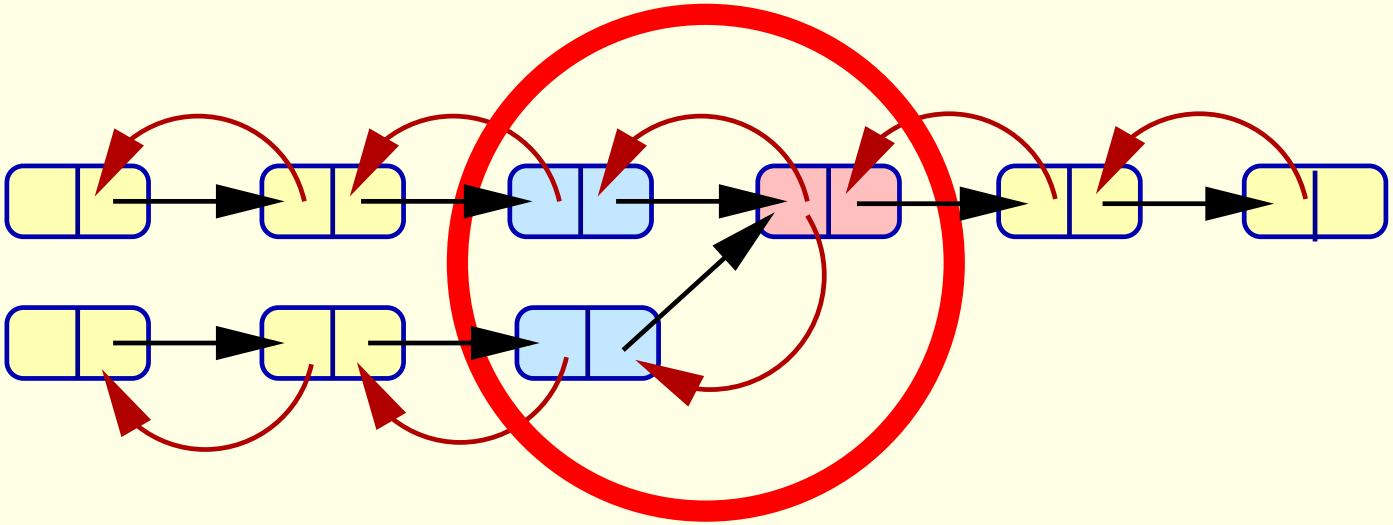
$(c \neq \text{null} \wedge c.\text{next} \neq \text{null} \rightarrow c.\text{next.prev} = c)$ $(c.\text{next} \neq \text{null} \wedge c.\text{next.next} \neq \text{null} \rightarrow c.\text{next.next.prev} = c.\text{next})$

$(d \neq \text{null} \wedge d.\text{next} \neq \text{null} \rightarrow d.\text{next.prev} = d)$ $(d.\text{next} \neq \text{null} \wedge d.\text{next.next} \neq \text{null} \rightarrow d.\text{next.next.prev} = d.\text{next})$

$$\wedge \quad c \neq \text{null} \wedge c.\text{next} \neq \text{null} \wedge d \neq \text{null} \wedge d.\text{next} \neq \text{null} \wedge c.\text{next} = d.\text{next} \wedge c \neq d \quad \models \quad \bot$$

Locality: Idea

Reasoning about doubly-linked lists cf. also [Necula, McPeak 2005]



$(c \neq \text{null} \wedge c.\text{next} \neq \text{null} \rightarrow c.\text{next.prev} = c)$ $(c.\text{next} \neq \text{null} \wedge c.\text{next.next} \neq \text{null} \rightarrow c.\text{next.next.prev} = c.\text{next})$

$(d \neq \text{null}$ $= d.\text{next})$

$\wedge$

Extensions which also take the **elements** of the list:

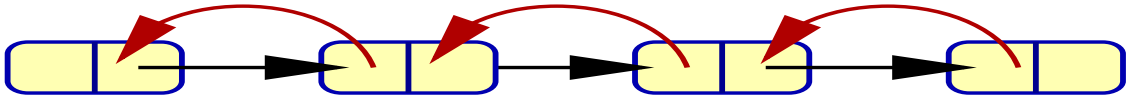
Can reasoning about lists be separated from reasoning about elements?

Reasoning in complex theories

Local reasoning in pointer structures

[McPeak, Necula 2005]

Example: doubly-linked lists; ordered elements



$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{next.prev} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{prev} \neq \text{null} \ \rightarrow\ p.\text{prev.next} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{info} \leq p.\text{next.info})$$

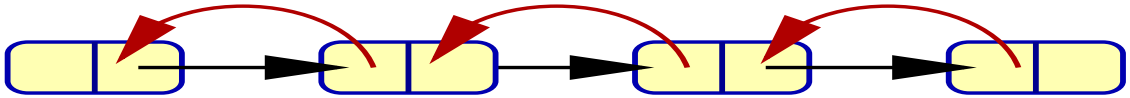
- pointer sort p, scalar sort s; pointer fields (p $\rightarrow$ p); scalar fields (p $\rightarrow$ s);
- axioms: $\forall p \ \ \mathcal{E} \wedge \mathcal{C}$; $\mathcal{E}$ contains disjunctions of pointer equalities
  $\mathcal{C}$ contains scalar constraints

Assumption: If $f_1(f_2(\ldots f_n(p)))$ occurs in axiom, the axiom also contains:
$$p{=}\text{null} \vee f_n(p){=}\text{null} \vee \cdots \vee f_2(\ldots f_n(p)))){=}\text{null}$$

Local reasoning in pointer structures

[McPeak, Necula 2005] **(stably local)**

Example: doubly-linked lists; ordered elements



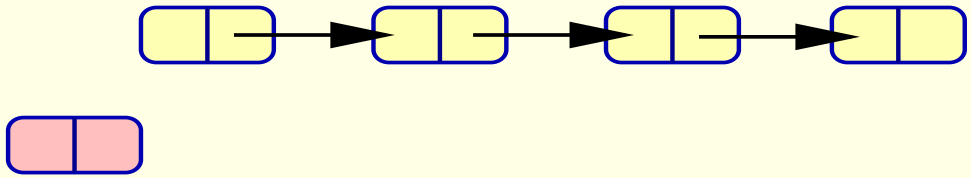
$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{next.prev} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{prev} \neq \text{null} \ \rightarrow\ p.\text{prev.next} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{info} \leq p.\text{next.info})$$

- pointer sort p, scalar sort s; pointer fields (p $\rightarrow$ p); scalar fields (p $\rightarrow$ s);
- axioms: $\forall p \ \ \mathcal{E} \wedge \mathcal{C}$; $\mathcal{E}$ contains disjunctions of pointer equalities
  $\mathcal{C}$ contains scalar constraints

Assumption: If $f_1(f_2(\ldots f_n(p)))$ occurs in axiom, the axiom also contains:
$$p{=}\text{null} \vee f_n(p){=}\text{null} \vee \cdots \vee f_2(\ldots f_n(p)))){=}\text{null}$$

Extension:[Ihlemann, Jacobs, Sofronie, TACAS 2008] chains of extensions
new fields defined according to a partition of the state space from old fields.

Example: List insertion



Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio}$

$c.\text{prio} = x$, $c.\text{next} = \text{null}$

for all $p \neq c$ **do**

if $p.\text{prio} \leq x$ **then if** $\text{First}(p)$ **then** $c.\text{next}' = p$, $\text{First}'(c)$, $\neg\text{First}'(p)$ **endif**; $p.\text{next}' = p.\text{next}$

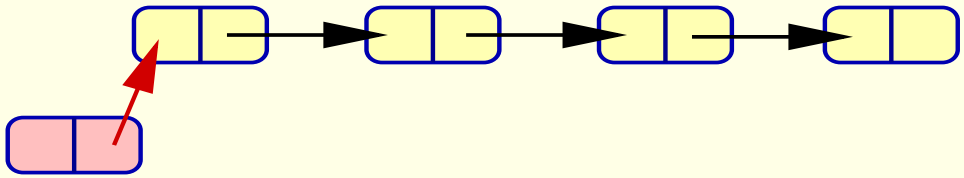
$p.\text{prio} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c$, $c.\text{next}' = \text{null}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} > x$ **then** $p.\text{next}' = p.\text{next}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} \leq x$ **then** $p.\text{next}' = c$, $c.\text{next}' = p.\text{next}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio}$

$c.\text{prio} = x$, $c.\text{next} = \text{null}$

for all $p \neq c$ **do**

if $p.\text{prio} \leq x$ **then if** $\text{First}(p)$ **then** $c.\text{next}' = p$, $\text{First}'(c)$, $\neg\text{First}'(p)$ **endif**; $p.\text{next}' = p.\text{next}$

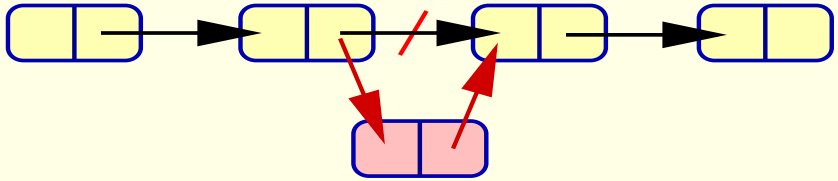
$p.\text{prio} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c$, $c.\text{next}' = \text{null}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} > x$ **then** $p.\text{next}' = p.\text{next}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} \leq x$ **then** $p.\text{next}' = c$, $c.\text{next}' = p.\text{next}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio}$

$c.\text{prio} = x$, $c.\text{next} = \text{null}$

for all $p \neq c$ **do**

if $p.\text{prio} \leq x$ **then if** $\text{First}(p)$ **then** $c.\text{next}' = p$, $\text{First}'(c)$, $\neg\text{First}'(p)$ **endif**; $p.\text{next}' = p.\text{next}$

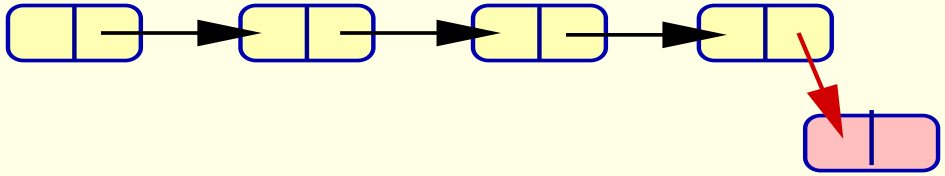
$p.\text{prio} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c$, $c.\text{next}' = \text{null}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} > x$ **then** $p.\text{next}' = p.\text{next}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} \leq x$ **then** $p.\text{next}' = c$, $c.\text{next}' = p.\text{next}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio}$

$c.\text{prio} = x$, $c.\text{next} = \text{null}$

for all $p \neq c$ **do**

if $p.\text{prio} \leq x$ **then if** $\text{First}(p)$ **then** $c.\text{next}' = p$, $\text{First}'(c)$, $\neg\text{First}'(p)$ **endif**; $p.\text{next}' = p.\text{next}$

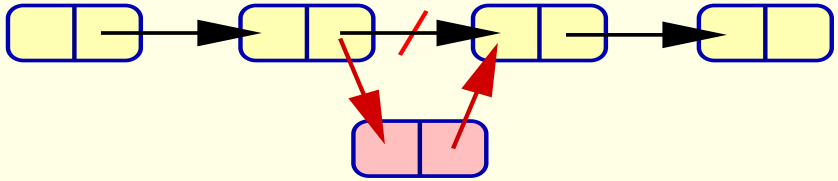
$p.\text{prio} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c$, $c.\text{next}' = \text{null}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} > x$ **then** $p.\text{next}' = p.\text{next}$

$p.\text{next} \neq \text{null} \wedge p.\text{next.prio} \leq x$ **then** $p.\text{next}' = c$, $c.\text{next}' = p.\text{next}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $\forall p(p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio})$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \text{First}(p) \rightarrow \text{next}'(c){=}p \wedge \text{First}'(c))$
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \text{First}(p) \rightarrow \text{next}'(p){=}\text{next}(p) \wedge \neg\text{First}'(p))$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \neg\text{First}(p) \rightarrow \text{next}'(p){=}\text{next}(p))$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){=}\text{null} \rightarrow \text{next}'(p){=}c$
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){=}\text{null} \rightarrow \text{next}'(c){=}\text{null})$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){\neq}\text{null} \wedge \text{prio}(\text{next}(p)){>}x \rightarrow \text{next}'(p){=}\text{next}(p))$

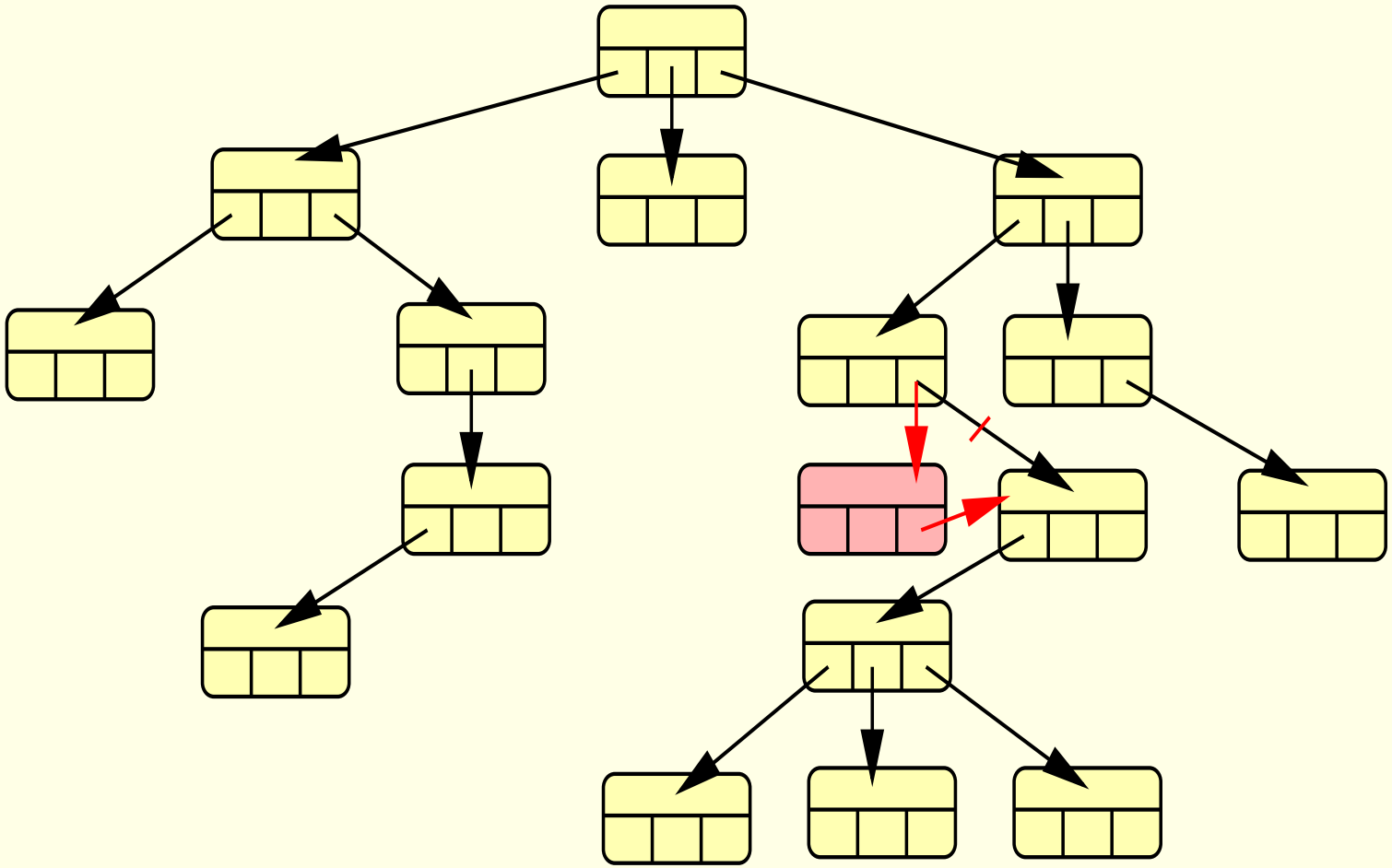
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){\neq}\text{null} \wedge \text{prio}(\text{next}(p)){\leq}x \rightarrow \text{next}'(p){=}c$
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){\neq}\text{null} \wedge \text{prio}(\text{next}(p)){\leq}x \rightarrow \text{next}'(c){=}\text{next}(p))$

Not in the fragment of [Necula,McPeak 2005], but proved to be local using our framework

---

To check: $\text{Sorted}(\text{next}, \text{prio}) \wedge \text{Update}(\text{next}, \text{next}') \wedge p_0.\text{next}' {\neq}\text{null} \wedge p_0.\text{prio} \not\geq p_0.\text{next}'.\text{prio} \models \bot$

More general pointer structures



Theories of arrays

- **Array property fragment** [Bradley,Manna,Sipma'06]

  $\exists$-closed Bool. comb. of array property formulae & QF formulae ($\exists\forall_i$)

  Array property $(\forall i)(\varphi_I(i) \rightarrow \varphi_V(i))$ Ex: $x \leq y \rightarrow a(x) \leq a(y)$

  φ_I: positive Boolean combination of $t \leq u$ or $t = u$, Ex: $x \leq y$
  where t, u ground index terms or variables $x \leq c$

  φ_V: any universally quantified i occurs in a direct array read; no nestings

- **Arrays + dimension; definedness** [Ghilardi,Nicolini,Ranise,Zucchelli'06]

Theories of arrays

- **Array property fragment** [Bradley,Manna,Sipma'06]

  $\exists$-closed Bool. comb. of array property formulae & QF formulae ($\exists\forall_i$)

  Array property $(\forall i)(\varphi_I(i) \rightarrow \varphi_V(i))$ Not: $x < y \rightarrow a(x) < a(y)$

  φ_I: positive Boolean combination of $t \leq u$ or $t = u$, Not: $x + 3 \leq 2 * y$

  where t, u ground index terms or variables $x + 1 \leq c$

  φ_V: any universally quantified i occurs in a direct array read; no nestings

- **Arrays + dimension; definedness** [Ghilardi,Nicolini,Ranise,Zucchelli'06]

[Ihlemann, Jacobs, Sofronie, TACAS 2008]

- both fragments above satisfy certain type of locality
- **Extension:** new arrays defined according to partition of the state space
- Some of the syntactic restrictions of [Bradley,Manna,Sipma'06] can be lifted

Example: Enabling processes

Parametric number m of processes

priorities: $p(1), \ldots, p(m) \in \mathbb{R}^+$

enabled: $a(1), \ldots, a(m) \in \{0, 1\}$

process nr.	1	2	3	4	5	6	7	···
enabled (0/1)	1	0	0	0	0	0	0	···
priority	0	7	3	1	9	4	2	···

process nr.	1	2	3	4	5	6	7	···
enabled (0/1)	0	0	0	0	1	0	0	···
priority	1	8	4	2	0	5	3	···

Update(a, p, a', p')

the process with maximal priority is enabled; priority set to x;

priorities of the waiting processes are increased by y

$$\forall i(1 \leq i \leq m \wedge \quad (\forall j(1 \leq j \leq m \wedge j \neq i \rightarrow p(i) > p(j))) \rightarrow a'(i) = 1 \wedge p'(i) = x)$$
$$\forall i(1 \leq i \leq m \wedge \neg(\forall j(1 \leq j \leq m \wedge j \neq i \rightarrow p(i) > p(j))) \rightarrow a'(i) = 0 \wedge p'(i) = p(i) + y)$$

Verification task: $\forall i, j(1 \leq i \neq j \leq m \rightarrow p(i) \neq p(j)) \wedge$ Update$(a, p, a', p') \wedge$
$(1 \leq c \neq d \leq m \wedge p'(c) = p'(d)) \models \bot$.

Assume priorities are all different. Do they remain different after update?
(determine x, y such that this is the case)

Conclusions

- Methodological framework for modular reasoning in complex theories: considerable advance compared with the state of the art

- Efficient implementation $\mapsto$ Carsten's talk

- Applications in the frame of the AVACS project $\mapsto$ Swen's talk