Decision Procedures in Verification

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Motivation

Long-term goal of research in computer science

- use computers as 'intelligent assistants' in

  e.g. mathematics, engineering (and other fields)

Main problem

- complex description of problems to be solved

  $\mapsto$ complex systems, complex encoding

Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- polynomials
- functions over numeric domains
- algebras

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

Example: Lipschitz functions

$$\mathbb{R} \cup (\mathsf{L}^{f}_{c,\lambda_1}) \cup (\mathsf{L}^{g}_{c,\lambda_2}) \models (\mathsf{L}^{f+g}_{c,(\lambda_1+\lambda_2)})$$

$(\mathsf{L}^{f}_{c,\lambda_1})$ $\forall x\ |f(x) - f(c)| \leq \lambda_1 \cdot |x - c|$

$(\mathsf{L}^{g}_{c,\lambda_2})$ $\forall x\ |g(x) - g(c)| \leq \lambda_2 \cdot |x - c|$

$(\mathsf{L}^{f+g}_{c,(\lambda_1+\lambda_2)})$ $\forall x\ |f(x)+g(x)-f(c)-g(c)| \leq (\lambda_1+\lambda_2) \cdot |x-c|$

Similar:
- free functions; (piecewise) monotone functions
- functions defined according to a partition of their domain of definition, ...

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- **reactive and hybrid systems**
  - safety / lifeness
- **programs**
  - correctness
  - termination

Infinite state systems (software, real time, hybrid)

- simulation/testing cannot guarantee absence of errors
  $\mapsto$ need symbolic methods

Solution: - Build 'formal model' of the system;
- Prove that properties are 'consequences of the model'

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- **reactive and hybrid systems**
  - safety / lifeness
- **programs**
  - correctness
  - termination

Theories
- numbers
- data types
- functions over numeric domains

Example: Does BUBBLESORT return a sorted array?

```
int [] BubbleSort(int[] a) {
  int i, j, t;
  for (i := |a| − 1; i > 0; i := i − 1) {
    for (j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]){t := a[j];
                           a[j] := a[j + 1];
                           a[j + 1] := t};
}} return a}
```

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- programs

correctness/termination
- reactive/hybrid systems

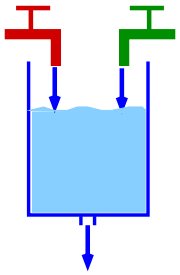
safety/lifeness
- cryptography

correctness crypt. prot.

Theories
- numbers
- data types
- functions over numeric domains

Controllers





Check:
- No overflow
- Substances in the right proportion
- If substances in wrong proportion, tank can be drained in $\leq$ 200s.

Determine values for parameters such that this is the case

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- programs

correctness/termination
- reactive/hybrid systems

safety/lifeness
- cryptography

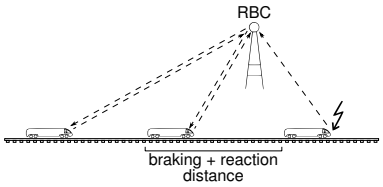
correctness crypt. prot.

Theories
- numbers
- data types
- functions over numeric domains

Controllers



Train controllers



- **Task:** check collision freeness

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- programs

correctness/termination
- reactive/hybrid systems

safety/lifeness
- cryptography

correctness crypt. prot.

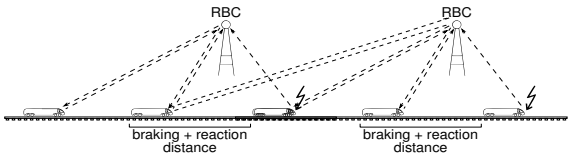
Theories
- numbers
- data types
- functions over numeric domains

Controllers

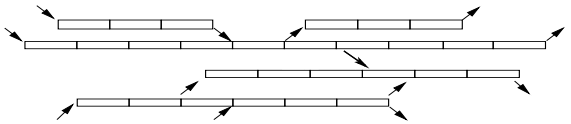


Two or more controllers

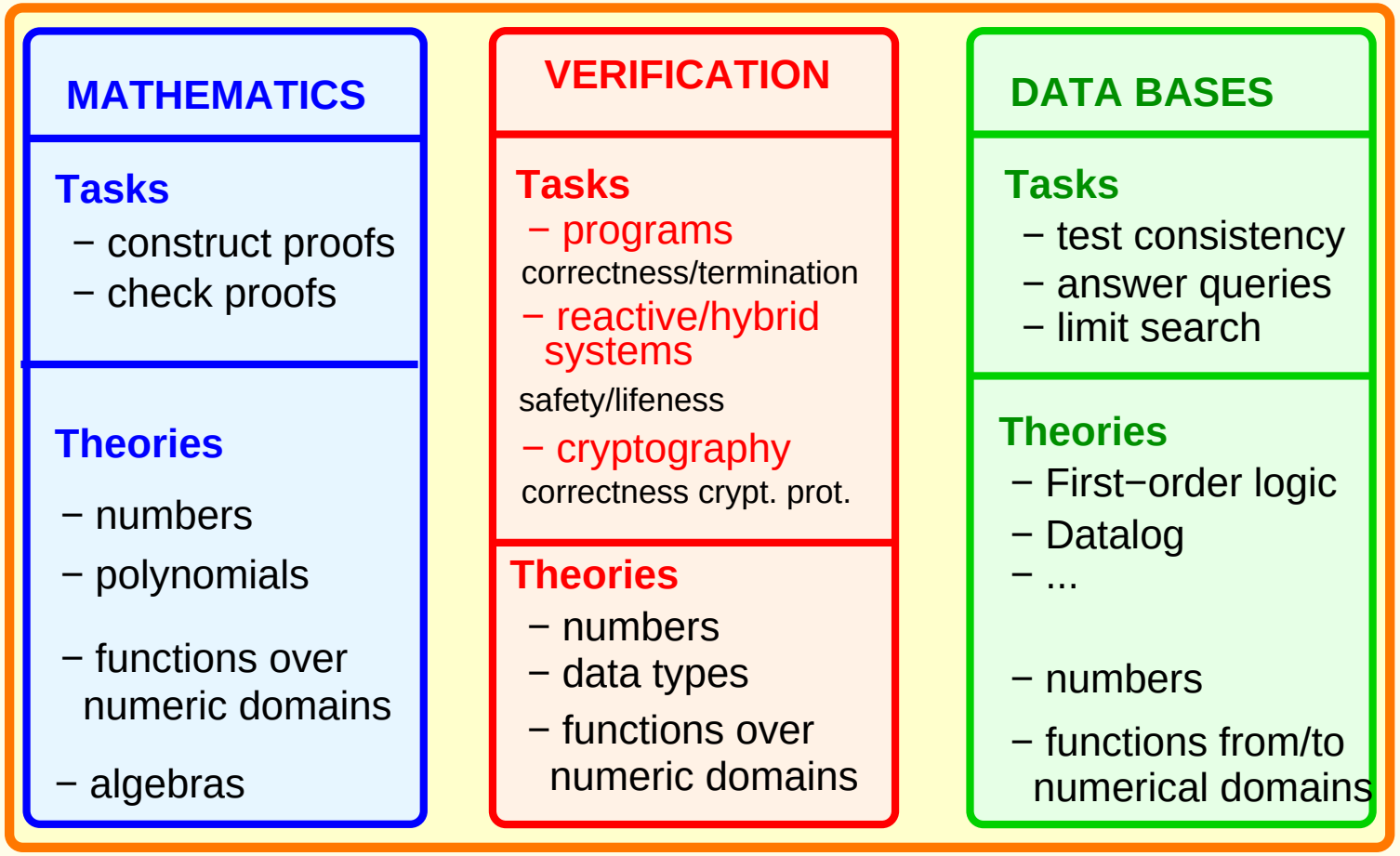
- non-disjoint sets of controlled trains
- synchronization for the control of common trains



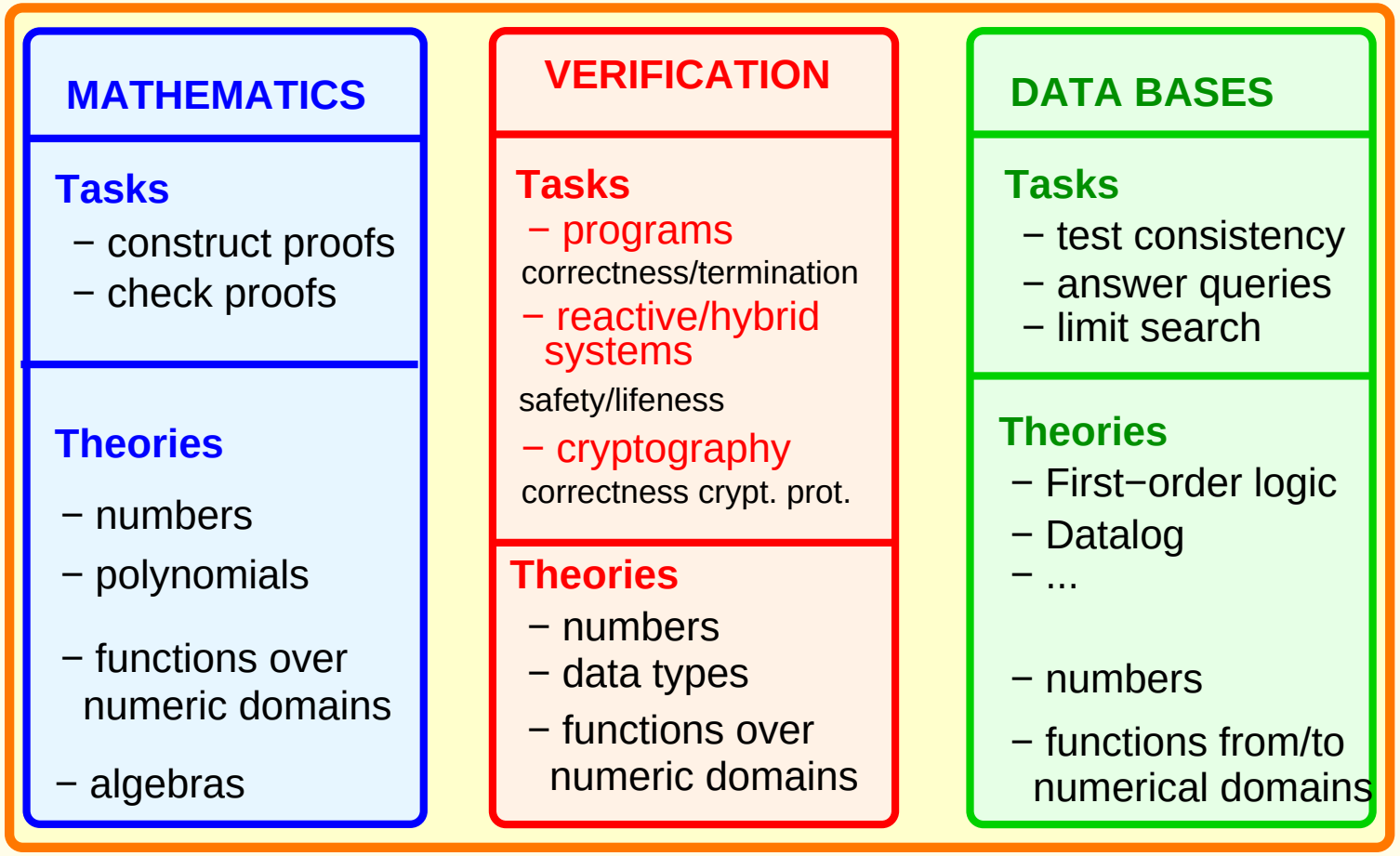
- complex track topology



Examples of application domains

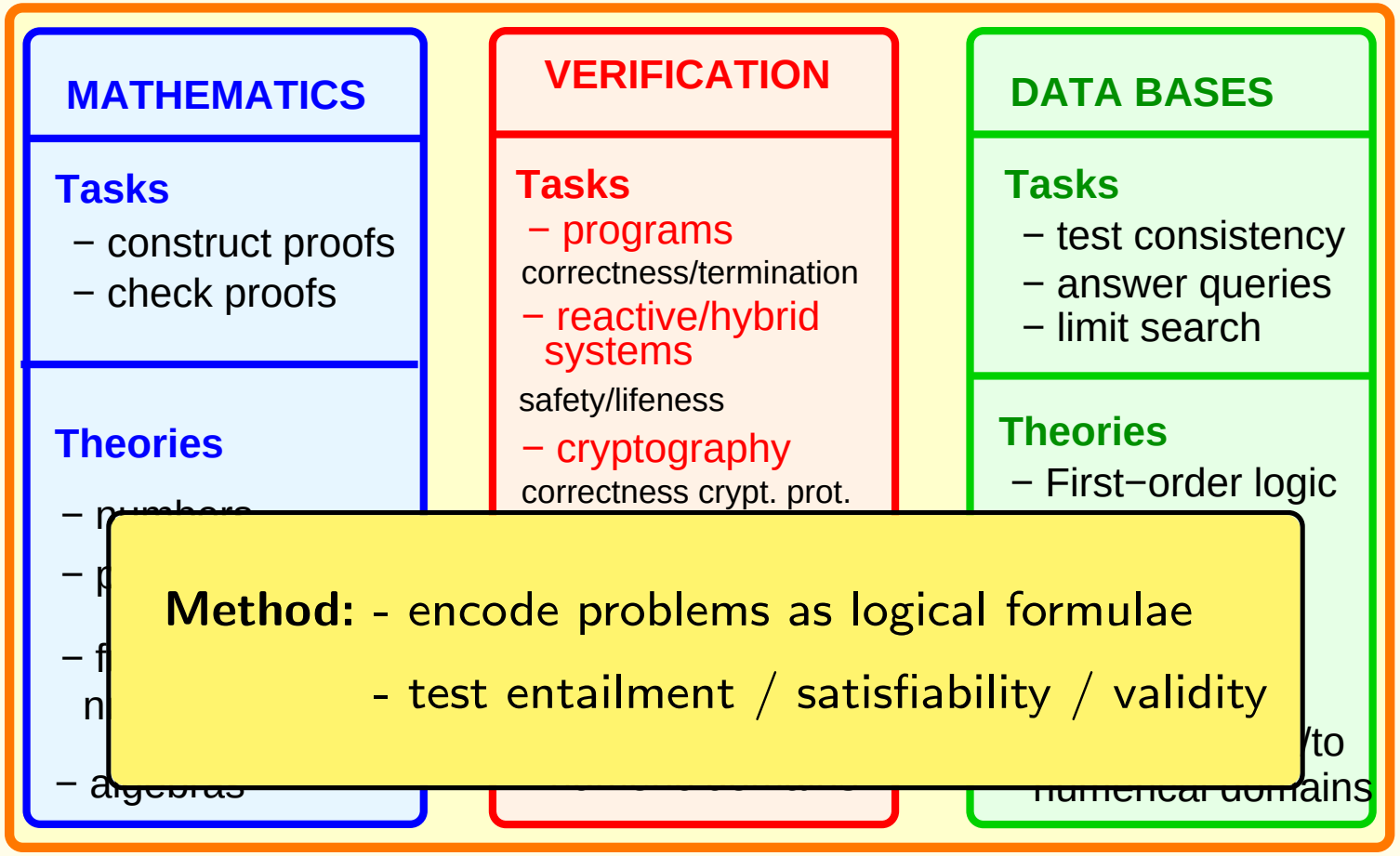


Examples of application domains



complex systems (MAS, reactive systems w. embedded software, databases)

Examples of application domains



complex systems (MAS, reactive systems w. embedded software, databases)

Problems and goals

– 1^{st} order logic is undecidable: cannot build an 'all-purpose' program

\+ often fragments of theories occurring in applications are decidable

– theories do not occur alone: need to consider combinations of theories

\+ often provers for the component theories can be combined efficiently

Problems and goals

– 1^{st} order logic is undecidable: cannot build an 'all-purpose' program

\+ often fragments of theories occurring in applications are decidable

– theories do not occur alone: need to consider combinations of theories

\+ often provers for the component theories can be combined efficiently

Important:

Identify theories (and extensions/combinations thereof) which are decidable (with low complexity) and relevant in applications

Overview

Plan of the lecture:

- **Reasoning in standard theories**
- **Reasoning in theory extensions**
- **Reasoning in combinations of theories**

Important: identify decidable/tractable fragments

Reasoning about standard datatypes

- **Numbers** - natural numbers, integers, reals, rationals

- **Data structures**
  - theories of lists
  - theory of acyclic lists
  - theory of arrays
  - theories of sets, multisets

- **Algebraic theories**
  - (total/partial) orderings
  - lattices, semilattices
  - distributive lattices
  - Boolean algebras
  - groups, rings, fields, ...

Reasoning in theory extensions

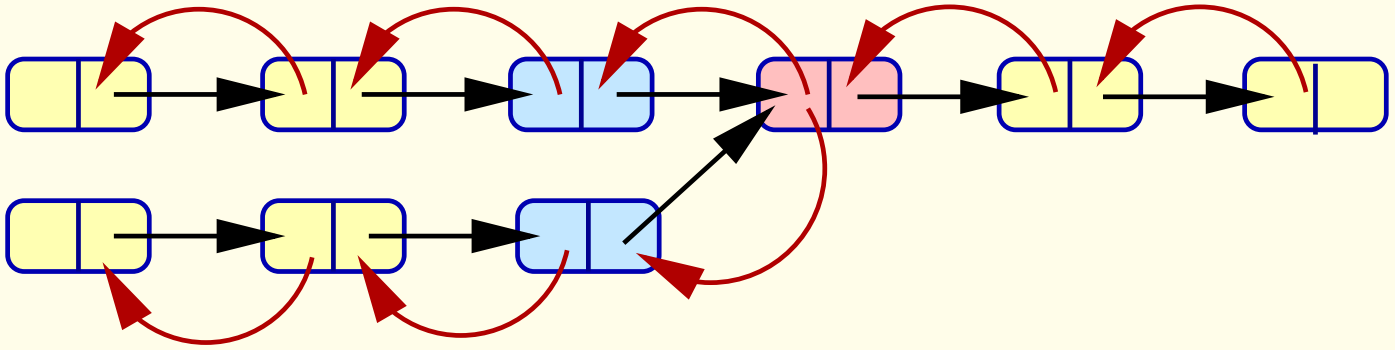
- **Numbers** - integers, reals, rationals

- **Data structures**
  - theories of lists of integers, reals, ...
  - theory of acyclic lists of integers, reals, ...
  - theory of arrays of integers, reals, ...
  - theories of sets of integers, reals, ...

  + functions (free, rec. def.) e.g : length, card

- **Algebraic theories**
  - (total/partial) orderings with monotone functions
  - lattices, semilattices with operators
  - distributive lattices with operators
  - Boolean algebras with operators
  - fields with operators

Example: A theory of doubly-linked lists

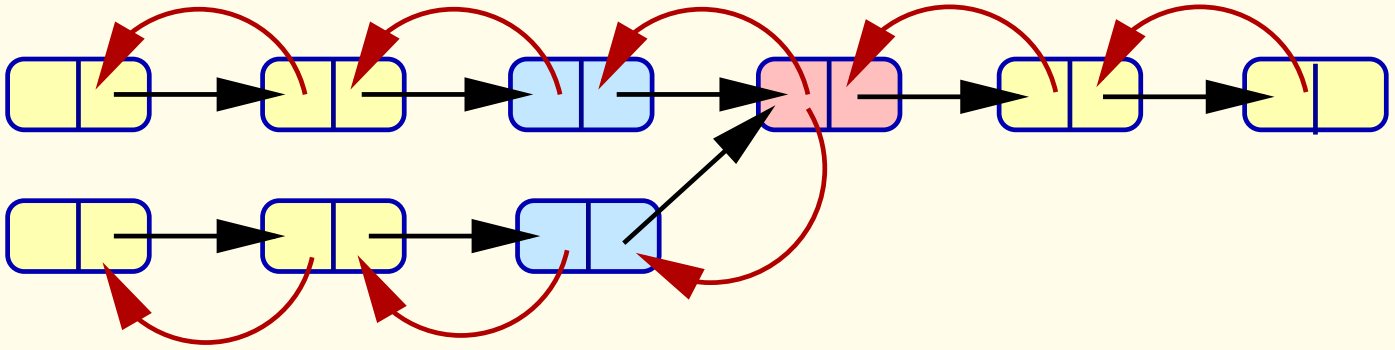


$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow \ p.\text{next.prev} = p)$$

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

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

Example: A theory of doubly-linked lists



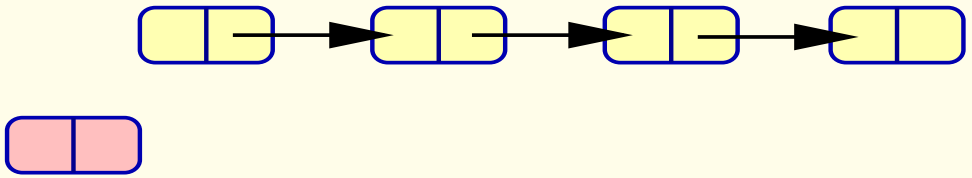
$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow \ p.\text{next.prev} = p)$$

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

Scalar fields + additional axioms:

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

Example: List insertion



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

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

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

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

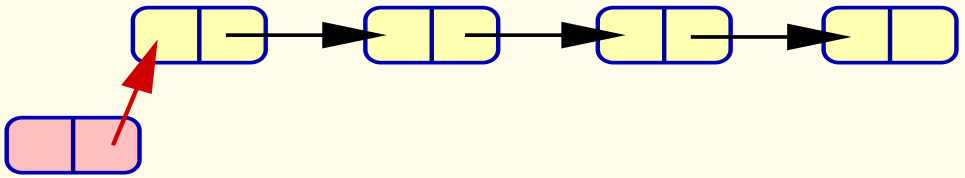
$p.\text{info} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c, c.\text{next}' = \text{null}$

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

$p.\text{next} \neq \text{null} \wedge p.\text{next.info} \leq x$ **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: $p.\text{next} \neq \text{null} \rightarrow p.\text{info} \geq p.\text{next.info}$

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

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

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

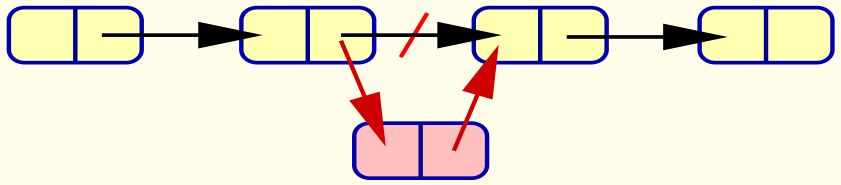
$p.\text{info} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c, c.\text{next}' = \text{null}$

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

$p.\text{next} \neq \text{null} \wedge p.\text{next.info} \leq x$ **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: $p.\text{next} \neq \text{null} \rightarrow p.\text{info} \geq p.\text{next.info}$

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

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

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

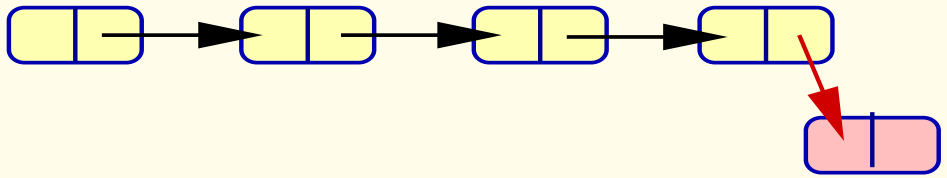
$p.\text{info} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c, c.\text{next}' = \text{null}$

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

$p.\text{next} \neq \text{null} \wedge p.\text{next.info} \leq x$ **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: $p.\text{next} \neq \text{null} \rightarrow p.\text{info} \geq p.\text{next.info}$

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

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

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

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

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

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

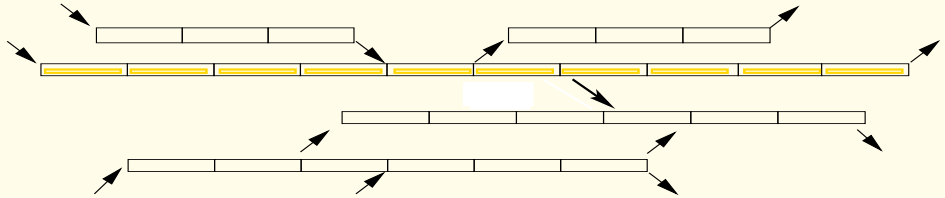
Verification task: After insertion list remains sorted

Applications

- Program verification
- More general verification problems

Model of a train controller

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, IFM 2010]



Assumptions:

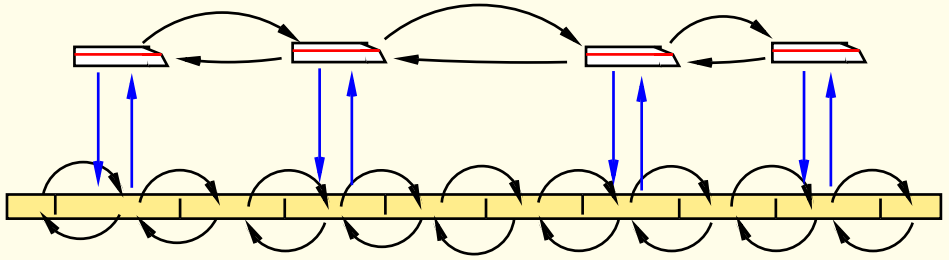
- No cycles
- in-degree (out-degree) of associated graph at most 2.

Data structures:

- 2-sorted pointers

  p_1: trains

  p_2: segments

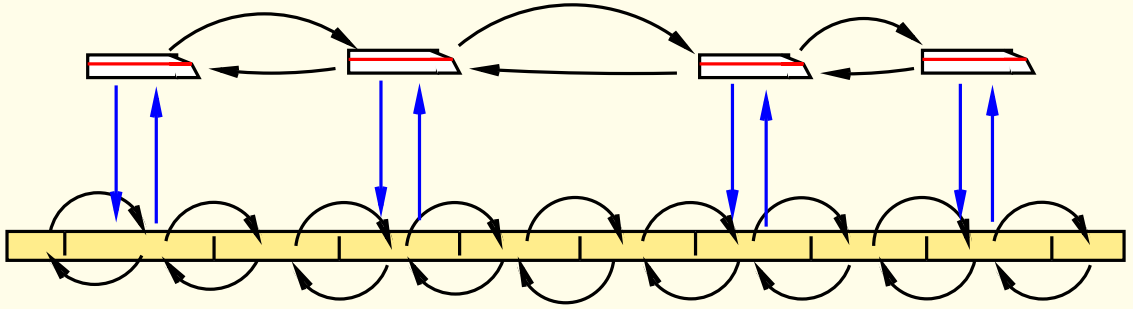


- scalar fields

  $\text{speed}: p_1 \to \mathbb{R}, \quad \text{id}_t : p_1 \to \mathbb{Z}$

  $\text{MaxSpeed}: p_2 \to \mathbb{R}, \quad \text{MaxBrakingDistance}: p_2 \to \mathbb{R}, \quad \text{id}_s : p_1 \to \mathbb{Z}$

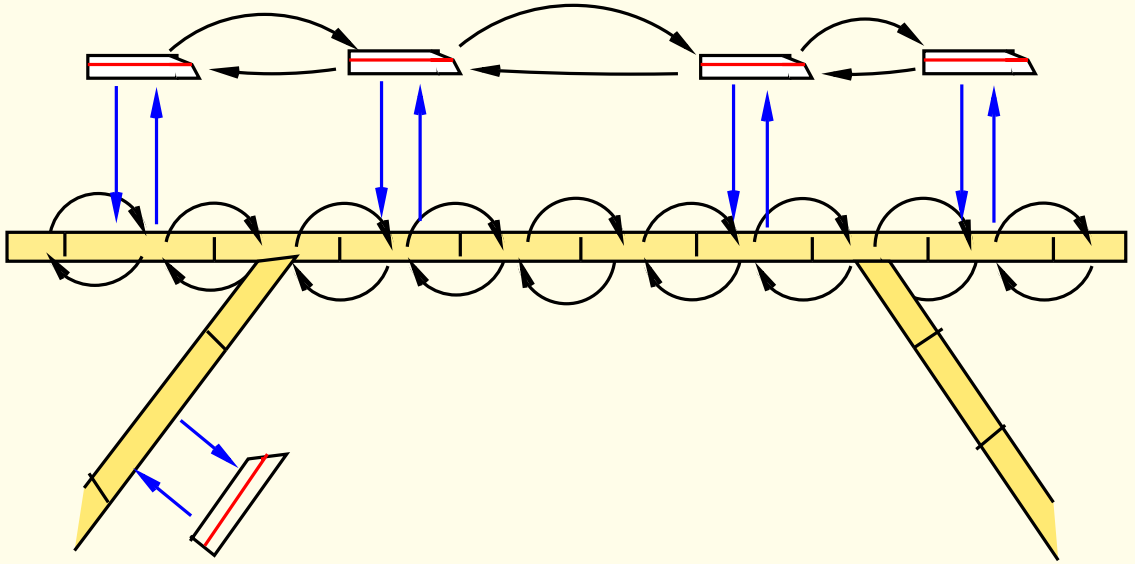
Incoming and outgoing trains



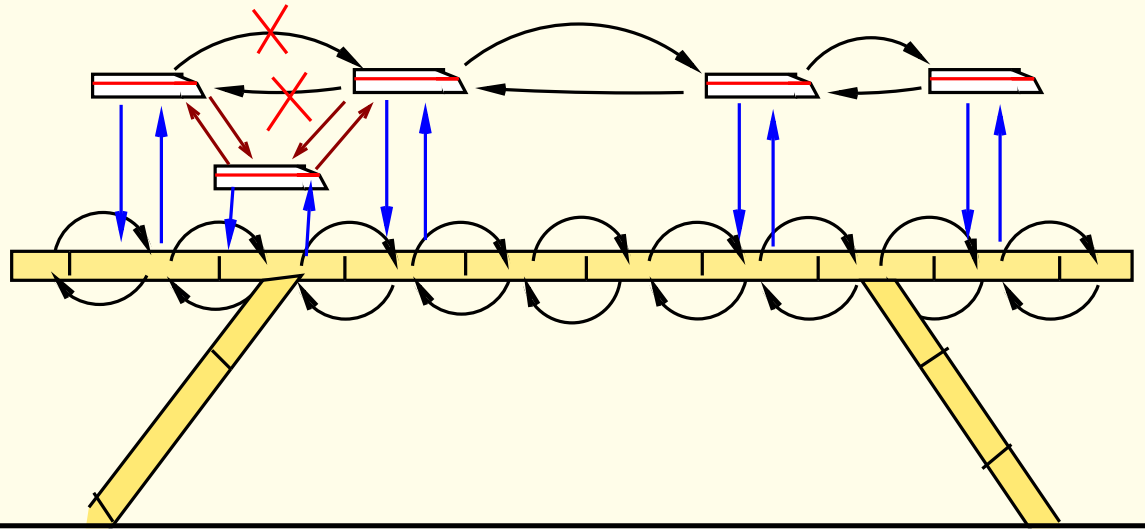
Example 1: Speed Update

$$\begin{aligned}
&\mathrm{pos}(t) < \mathrm{length}(\mathrm{segm}(t)) - d \;\rightarrow\; 0 \leq \mathrm{spd}'(t) \leq \mathrm{lmax}(\mathrm{segm}(t)) \\
&\mathrm{pos}(t) \geq \mathrm{length}(\mathrm{segm}(t)) - d \;\wedge\; \mathrm{alloc}(\mathrm{next}_s(\mathrm{segm}(t))) = \mathrm{tid}(t) \\
&\qquad\qquad \rightarrow\; 0 \leq \mathrm{spd}'(t) \leq \min(\mathrm{lmax}(\mathrm{segm}(t)), \mathrm{lmax}(\mathrm{next}_s(\mathrm{segm}(t)))) \\
&\mathrm{pos}(t) \geq \mathrm{length}(\mathrm{segm}(t)) - d \;\wedge\; \mathrm{alloc}(\mathrm{next}_s(\mathrm{segm}(t))) \neq \mathrm{tid}(t) \\
&\qquad\qquad \rightarrow\; \mathrm{spd}'(t) = \max(\mathrm{spd}(t) - \mathrm{decmax}, 0)
\end{aligned}$$

Incoming and outgoing trains



Incoming and outgoing trains



Example 2: Enter Update (also updates for segm', spd', pos', train')

Assume: $s_1 \neq \text{null}_s$, $t_1 \neq \text{null}_t$, $\text{train}(s) \neq t_1$, $\text{alloc}(s_1) = \text{idt}(t_1)$

$t{\neq}t_1, \text{ids}(\text{segm}(t)){<}\text{ids}(s_1), \text{next}_t(t){=}\text{null}_t, \text{alloc}(s_1){=}\text{tid}(t_1) \rightarrow \text{next}'(t){=}t_1 \wedge \text{next}'(t_1){=}\text{null}_t$

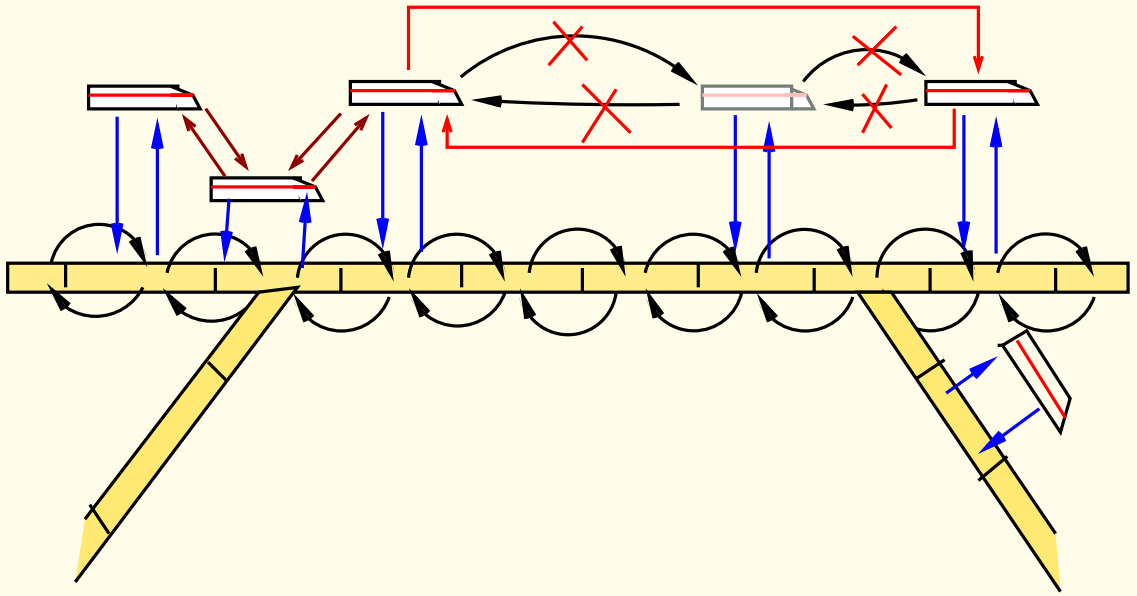
$t{\neq}t_1, \text{ids}(\text{segm}(t)){<}\text{ids}(s_1), \text{alloc}(s_1){=}\text{tid}(t_1), \text{next}_t(t){\neq}\text{null}_t, \text{ids}(\text{segm}(\text{next}_t(t))){\leq}\text{ids}(s_1)$

$\rightarrow \text{next}'(t){=}\text{next}_t(t)$

...

$t{\neq}t_1, \text{ids}(\text{segm}(t)){\geq}\text{ids}(s_1) \rightarrow \text{next}'(t){=}\text{next}_t(t)$

Incoming and outgoing trains



Safety property

Safety property we want to prove: no two trains ever occupy the same track segment:

$$(\text{Safe}) := \forall t_1, t_2 \;\; \text{segm}(t_1) = \text{segm}(t2) \rightarrow t_1 = t_2$$

We need to be able to efficiently reason in combinations of theories.

Example 2

Example: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
       for (j := 0; j < i; j := j + 1) {
          if (a[j] > a[j + 1]){t := a[j];
                               a[j] := a[j + 1];
                               a[j + 1] := t};
}} return a}
```

Example 2

$-1 \leq i < |a| \wedge$
partitioned$(a, 0, i, i+1, |a|-1) \wedge$
sorted$(a, i, |a|-1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$
partitioned$(a, 0, i, i+1, |a|-1) \wedge$
sorted$(a, i, |a|-1)$
partinioned$(a, 0, j-1, j, j)$

Example: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                                 a[j] := a[j + 1];
                                 a[j + 1] := t};
}} return a}
```

Generate verification conditions and prove that they are valid

Predicates:

- sorted(a, l, u): $\forall i, j (l \leq i \leq j \leq u \rightarrow a[i] \leq a[j])$
- partitioned(a, l_1, u_1, l_2, u_2): $\forall i, j (l_1 \leq i \leq u_1 \leq l_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j])$

Example 2

$-1 \leq i < |a| \wedge$ $\boxed{C_1(a)}$

$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$

$\text{sorted}(a, i, |a|-1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$ $\boxed{C_2(a)}$

$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$

$\text{sorted}(a, i, |a|-1)$

$\text{partinioned}(a, 0, j-1, j, j)$

Example: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                                 a[j] := a[j + 1];
                                 a[j + 1] := t};
}} return a}
```

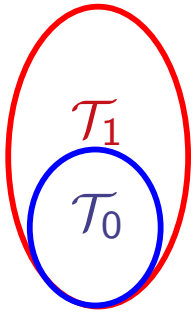
Generate verification conditions and prove that they are valid

$C_2(a) \wedge \text{Update}(a, a') \rightarrow C_2(a')$

Motivation

Modular (i.e. black-box) composition of decision procedures is highly desirable – for saving time and resources.

Idea



Hierarchic Reasoning

$\mathcal{T}_1$: Σ_1-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_1$ $\Sigma_0 \subset \Sigma_1$

$\mathcal{T}_0$: Σ_0-theory.

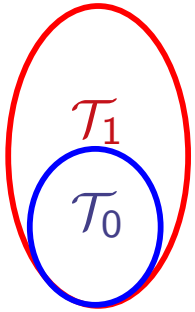
Example:

$f : \mathbb{R} \to \mathbb{R}$ mon.

$\mathbb{R}$

Can we use a prover for $\mathcal{T}_0$ as a blackbox to prove theorems in $\mathcal{T}_1$?

Idea



Hierarchic Reasoning

$\mathcal{T}_1$: Σ_1-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_1$ $\Sigma_0 \subset \Sigma_1$

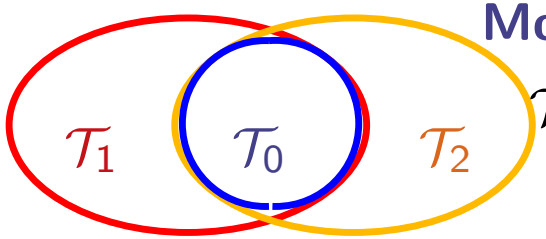
$\mathcal{T}_0$: Σ_0-theory.

Example:

$f : \mathbb{R} \to \mathbb{R}$ mon.

$\mathbb{R}$

Can we use a prover for $\mathcal{T}_0$ as a blackbox to prove theorems in $\mathcal{T}_1$?



Modular Reasoning

$\mathcal{T}_0$: Σ_0-theory.

$\mathcal{T}_i$: Σ_i-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_i$ $\Sigma_0 \subseteq \Sigma_i$.

Example: **arrays**$(\mathbb{Z}, \mathbb{R})$

lists$(\mathbb{R})$ $\cup$ **arrays**$(\mathbb{Z}, \mathbb{R})$

Can we use provers for $\mathcal{T}_1, \mathcal{T}_2$ as blackboxes to prove theorems in $\mathcal{T}_1 \cup \mathcal{T}_2$?

Which information needs to be exchanged between the provers?

Structure

- **Reasoning in standard theories**

  Preliminaries: Logic, theories, models

  Decidable logical theories and theory fragments

  Tractability

- **Reasoning in combinations of theories**

  disjoint signature

  non-disjoint signature

- **Theory extensions**

- **Applications in Verification**

Reasoning in standard theories

- **Propositional logic and first-order logic** (reminder)
  - Syntax, Semantics, Entailment, Validity, Satisfiability
  - Theories, Models
  - Deduction in propositional logic (DPLL, resolution)
  - Deduction in first-order logic (the resolution principle)
- **Decidability and undecidability results**
  - Undecidability of FOL/Some decidable fragments
  - Deduction problems
- **Reasoning in numerical domains** (a crash course)

Reasoning in complex theories

- **Reasoning in combinations of theories**
  - combinations of theories with disjoint signatures
  - combinations of theories with non-disjoint signatures (main idea)
- **Theory extensions**
- **Applications**
  - Decision procedures for data structures:
    - arrays
    - pointer structures
    - sets (with cardinalities)
    - recursive data structures + recursive functions
  - Applications in verification

Literature

Logic:

Any book on logic (propositional and first-order logic)

- The slides of the logic lecture
- Schöning: Logik für Informatiker, Spektrum
- Fitting: First-Order Logic and Automated Theorem Proving, Springer

Literature

Decision procedures:

- Aaron Bradley and Zohar Manna: The Calculus of Computation: Decision Procedures with Applications to Verification. Springer, 2007.

- Daniel Kroening and Ofer Strichman: Decision Procedures: An algorithmic point of view. Springer, 2007.

Viorica Sofronie-Stokkermans: Reasoning in complex theories. Lecture notes/slides

Semesterapparat at the library