Decision Procedures in Verification

Winter Semester 2013/14

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
  ↦ 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

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

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

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

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

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

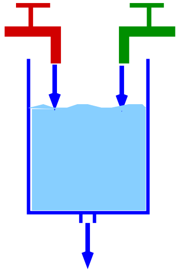
- cryptography

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

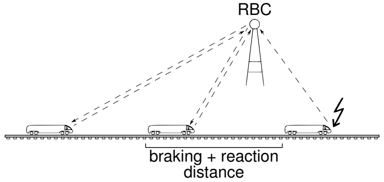
correctness crypt. prot.

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.

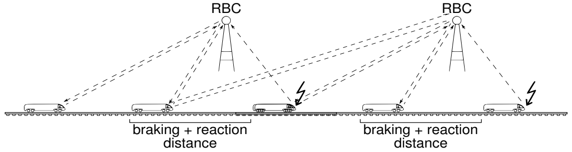
Theories

- numbers
- data types
- functions over numeric domains

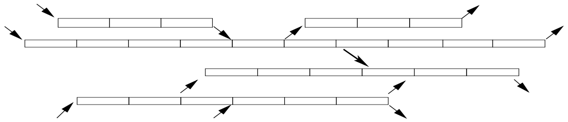
Controllers

Two or more controllers

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



- complex track topology



Examples of application domains

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

DATA BASES

Tasks

- test consistency
- answer queries
- limit search

Theories

- First−order logic
- Datalog
- ...
- numbers
- functions from/to numerical domains

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

DATA BASES

Tasks

- test consistency
- answer queries
- limit search

Theories

- First−order logic
- Datalog
- ...

- numbers
- functions from/to numerical domains

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

Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- p
- f
  n
- algebras

VERIFICATION

Tasks

- programs
  correctness/termination
- reactive/hybrid systems
  safety/lifeness
- cryptography
  correctness crypt. prot.

DATA BASES

Tasks

- test consistency
- answer queries
- limit search

Theories

- First−order logic

/to
numerical domains

Method: - encode problems as logical formulae

- test entailment / satisfiability / validity

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

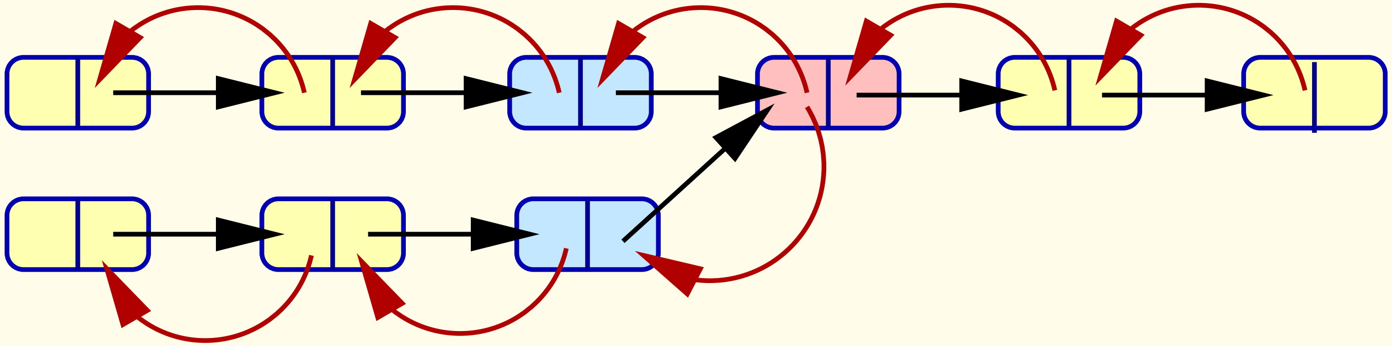
Reasoning about standard datatypes

- **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

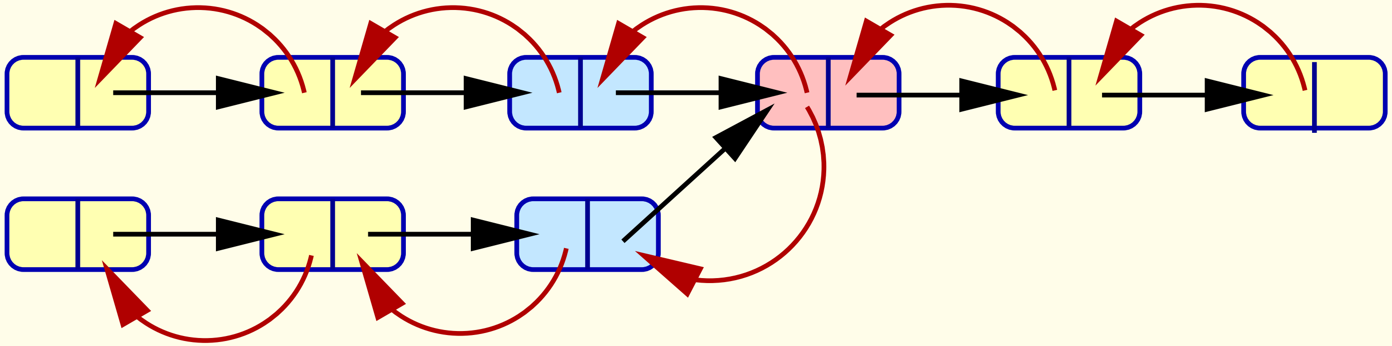


$$\forall p\ (p \neq \mathrm{null} \wedge p.\mathrm{next} \neq \mathrm{null} \ \rightarrow\ p.\mathrm{next}.\mathrm{prev} = p)$$

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

$$\wedge\ \ c{\neq}\mathrm{null} \wedge c.\mathrm{next}{\neq}\mathrm{null} \wedge d{\neq}\mathrm{null} \wedge d.\mathrm{next}{\neq}\mathrm{null} \wedge c.\mathrm{next}{=}d.\mathrm{next} \wedge c \neq d \quad \models \quad \perp$$

Example: A theory of doubly-linked lists



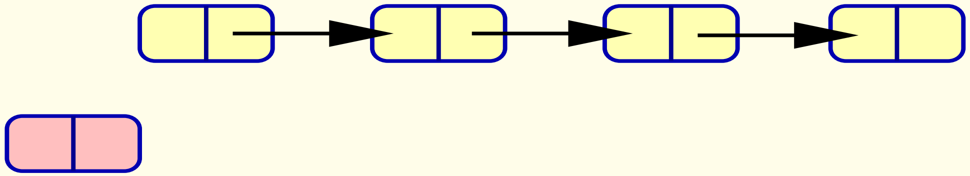
$$\forall p\ (p \neq \mathrm{null} \wedge p.\mathrm{next} \neq \mathrm{null} \ \rightarrow\ p.\mathrm{next}.\mathrm{prev} = p)$$

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

Scalar fields + additional axioms:

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

Example: List insertion

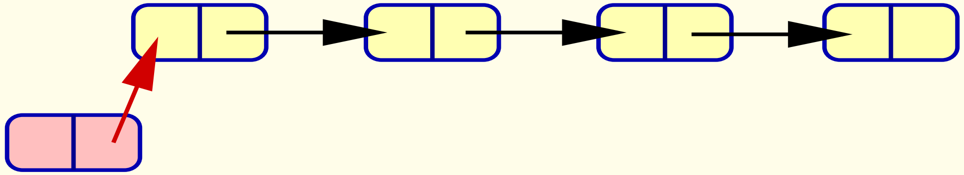


Initially list is sorted: $p.\mathrm{next} \neq \mathrm{null} \rightarrow p.\mathrm{info} \geq p.\mathrm{next}.\mathrm{info}$

```
$c.\mathrm{info} = x$, $c.\mathrm{next} = \mathrm{null}$
for all $p \neq c$ do
if $p.\mathrm{info} \leq x$ then if $\mathrm{First}(p)$ then $c.\mathrm{next}' = p$, $\mathrm{First}'(c)$, $\neg\mathrm{First}'(p)$ endif; $p.\mathrm{next}' = p.\mathrm{next}$
   $p.\mathrm{info} > x$ then   case $p.\mathrm{next} = \mathrm{null}$ then $p.\mathrm{next}' := c$, $c.\mathrm{next}' = \mathrm{null}$
                                $p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{info} > x$ then $p.\mathrm{next}' = p.\mathrm{next}$
                                $p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{info} \leq x$ then $p.\mathrm{next}' = c$, $c.\mathrm{next}' = p.\mathrm{next}$
```

Verification task: After insertion list remains sorted

Example: List insertion

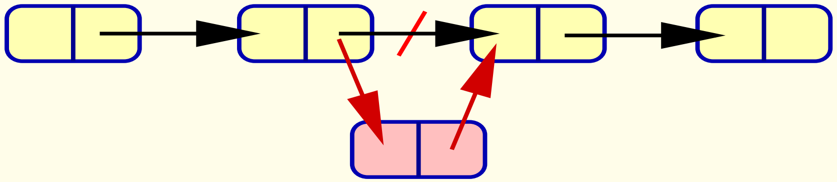


Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{info} \geq p.\text{next}.\text{info}$

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

Verification task: After insertion list remains sorted

Example: List insertion

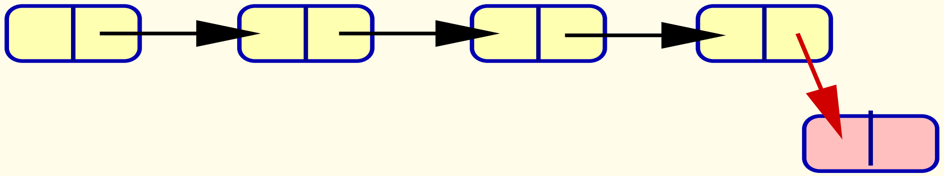


Initially list is sorted: $p.\mathrm{next} \neq \mathrm{null} \rightarrow p.\mathrm{info} \geq p.\mathrm{next}.\mathrm{info}$

```
$c.\mathrm{info} = x$, $c.\mathrm{next} = \mathrm{null}$
for all $p \neq c$ do
if $p.\mathrm{info} \leq x$ then if $\mathrm{First}(p)$ then $c.\mathrm{next}' = p$, $\mathrm{First}'(c)$, $\neg\mathrm{First}'(p)$ endif; $p.\mathrm{next}' = p.\mathrm{next}$
   $p.\mathrm{info} > x$ then   case $p.\mathrm{next} = \mathrm{null}$ then $p.\mathrm{next}' := c$, $c.\mathrm{next}' = \mathrm{null}$
                                $p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{info} > x$ then $p.\mathrm{next}' = p.\mathrm{next}$
                                $p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{info} \leq x$ then $p.\mathrm{next}' = c$, $c.\mathrm{next}' = p.\mathrm{next}$
```

Verification task: After insertion list remains sorted

Example: List insertion



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

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

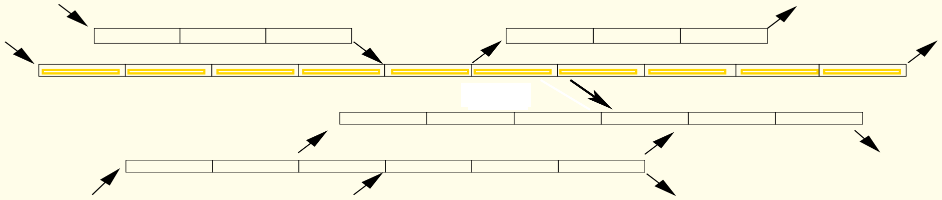
Verification task: After insertion list remains sorted

Applications

- Program verification
- More general verification problems

Model of a train controller

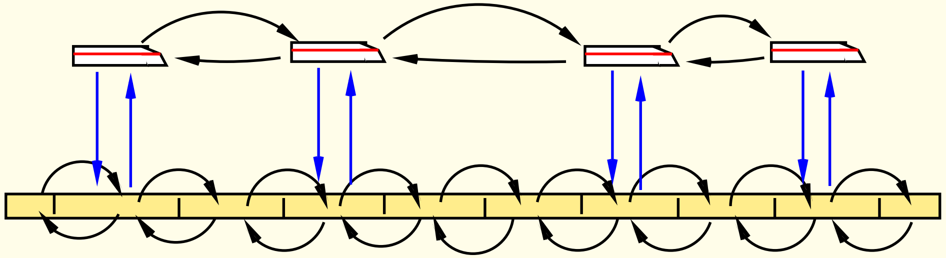
- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, IFM 2010]



Assumptions:

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

Data structures:



- 2-sorted pointers

  p_1: trains

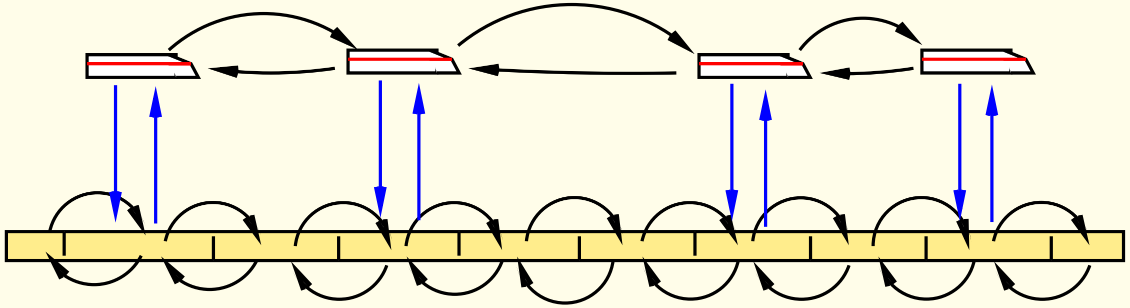
  p_2: segments

- scalar fields

  speed:$p_1 \rightarrow \mathbb{R}$, id$_t$:$p_1 \rightarrow \mathbb{Z}$

  MaxSpeed:$p_2 \rightarrow \mathbb{R}$, MaxBrakingDistance:$p_2 \rightarrow \mathbb{R}$, id$_s$:$p_1 \rightarrow \mathbb{Z}$

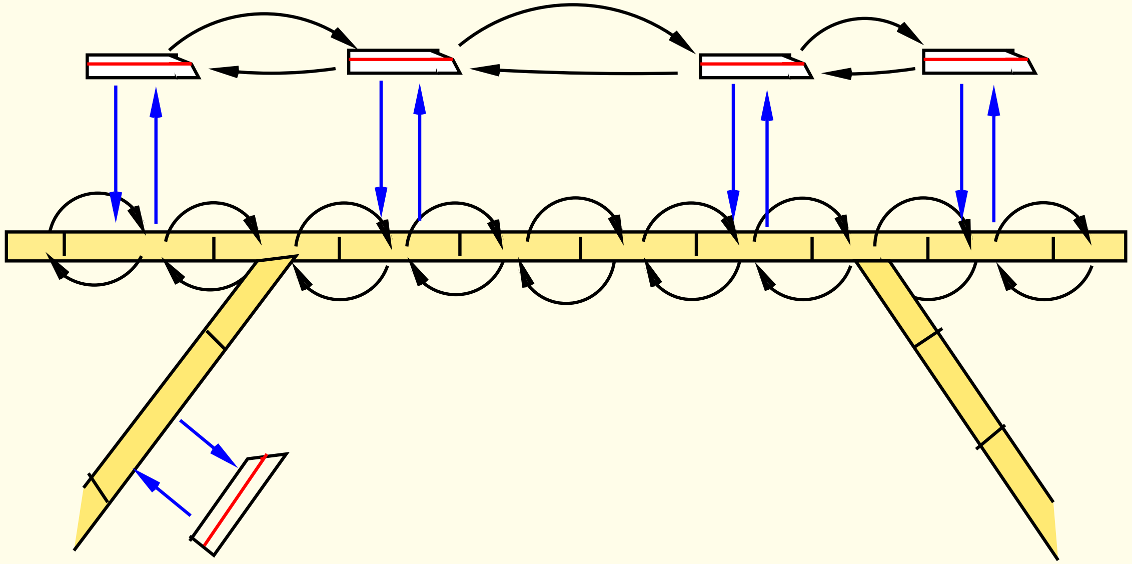
Incoming and outgoing trains



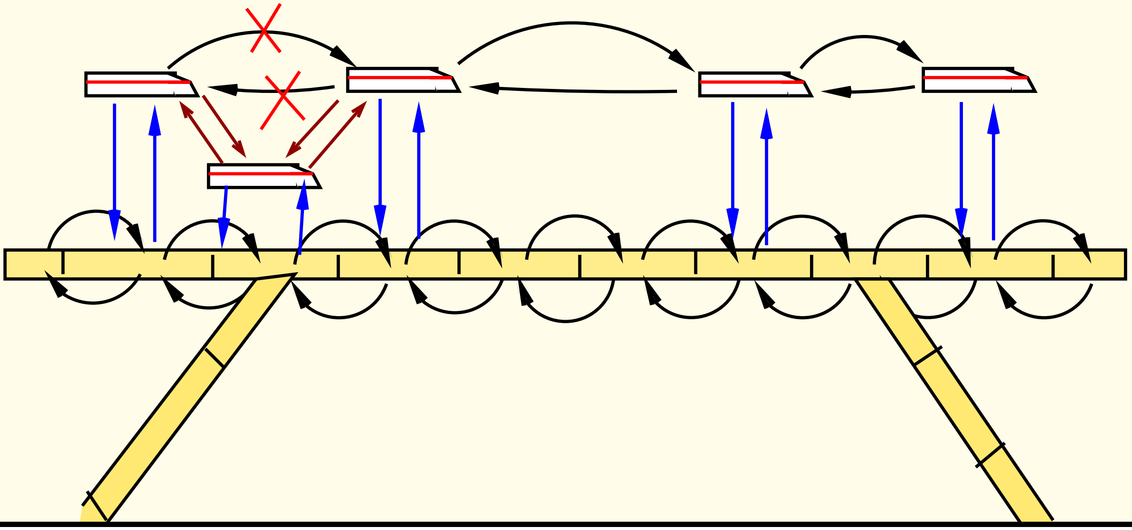
Example 1: Speed Update

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

Incoming and outgoing trains



Incoming and outgoing trains



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

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

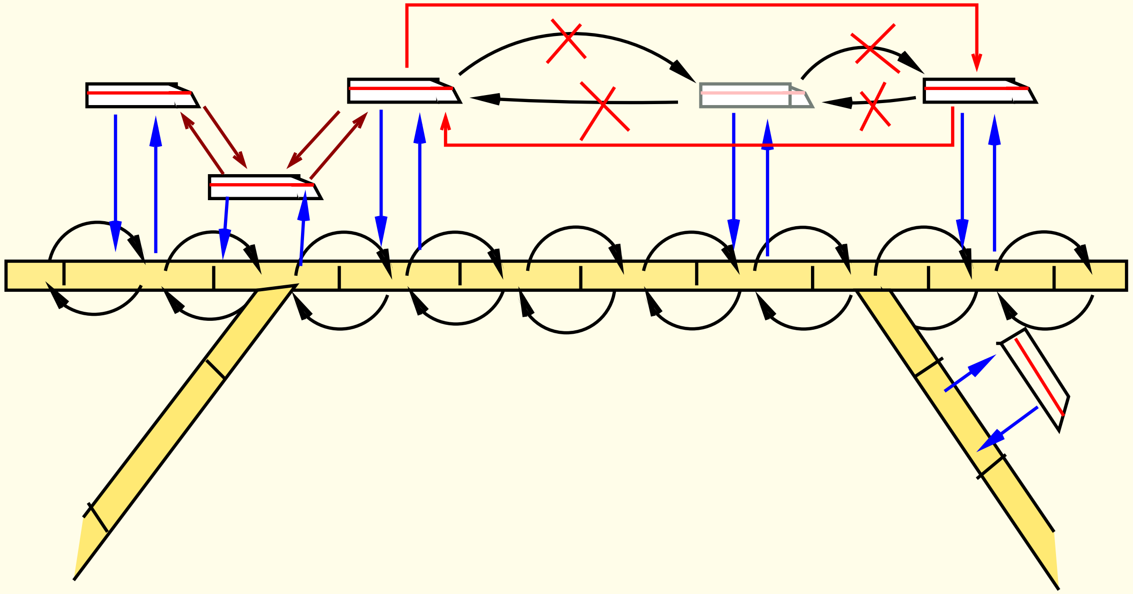
$t{\neq}t_1, \text{ids}(\text{segm}(t)){<}\text{ids}(s_1), \text{next}_t(t){=}\text{null}_t, \text{alloc}(s_1){=}\text{tid}(t_1) \rightarrow \text{next}'(t){=}t_1 \wedge \text{next}'(t_1){=}\text{null}_t$

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

...

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

Incoming and outgoing trains



Safety property

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

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

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

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 $a$}
```

$-1 \leq i < |a| \wedge$
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$(a, 0, j - 1, j, j)$

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 $a$}
```

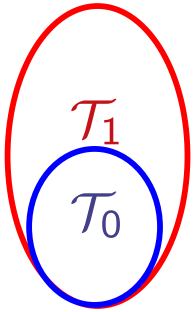
Generate verification conditions and prove that they are valid

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

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.

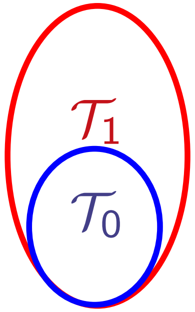
Example:

$f : \mathbb{R} \rightarrow \mathbb{R}$ 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; $\mathcal{T}_0 \subseteq \mathcal{T}_1$ $\Sigma_0 \subset \Sigma_1$

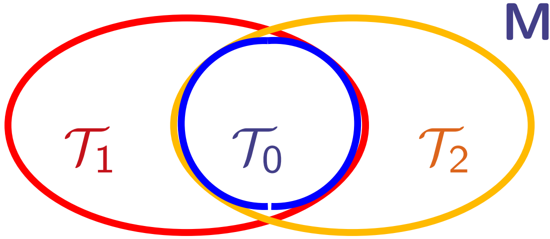
$\mathcal{T}_0$: Σ_0-theory.

Example:

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

$\mathbb{R}$

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



Modular Reasoning

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

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

$\mathcal{T}_0$: Σ_0-theory.

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

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)

- Schöning: Logik für Informatiker, Spektrum
- Fitting: First-Order Logic and Automated Theorem Proving, Springer
- Slides and suggested readings for the lecture “Automated Reasoning” (Saarbruecken, Summer Semester 2004), available online at:
  http://www.mpi-sb.mpg.de/ uwe/lehre/autreas/readings.html

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; will be available online soon.

Semesterapparat at the library

Organization

Organization: 2h Lecture + 2h Exercises

Time:

Mondays:	Lecture 10:00-12:00, Room C208
Wednesday:	Exercise 10:00-12:00, Room C208

Will probably be changed. Doodle for finding new slots:
`http://www.doodle.com/59xx3muz59rcra56`

website of the lecture:
`http://www.uni-koblenz.de/~sofronie/lecture-dp-ws-2013/`

Homework
- will be avaliable online after the lecture (at latest on Tuesday)
- due one-two days before the exercises (details will be announced)

Exam: form (oral/written): to be decided