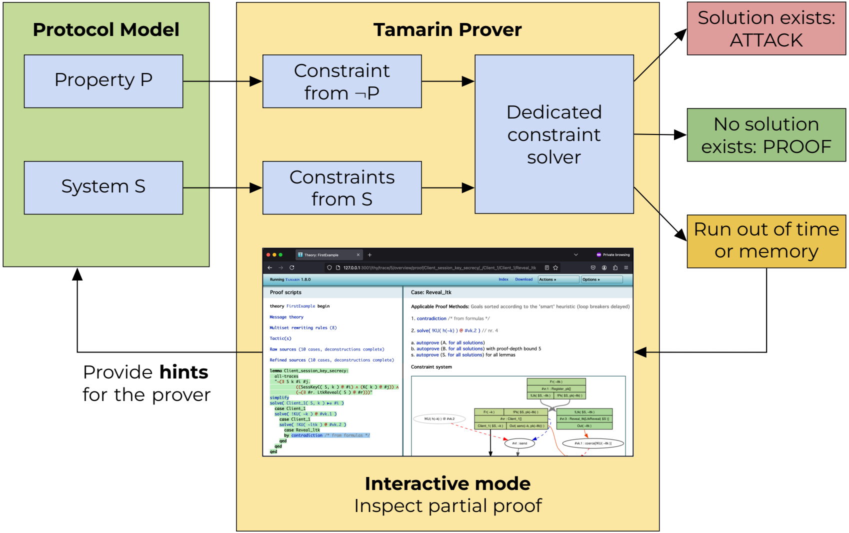
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

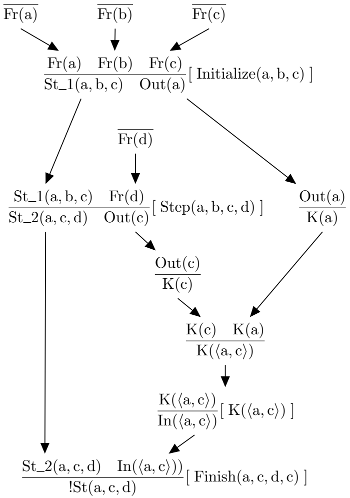
Lecture 5: Verification Theory (Part 2)

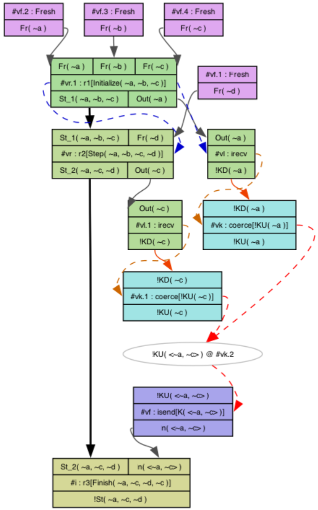
Aleksi Peltonen | 25.11.2024

Recap: Tamarin workflow



Recap: Dependency graphs





This lecture

Constraint Solving

Proof Methods

Constraint Solving

Constraint solving

Given a set of **rules** *R* and a **property** *P* :

- If the property is *all-traces* (the default):
  - Consider a set of constraints that represent (*R* and $\neg$*P*)
  - No solution is proof of *P*
  - Solutions are **counterexamples**
- If the property is *exists-trace*:
  - Consider a set of constraints that represent (*R* and *P*)
  - No solution means that *P* does not hold
  - Solutions are witnesses that *P* holds for some trace

Constraint solving

1. **Precomputation for rules**
   - Using static analysis, try to infer which rules must precede others
   - Compute *sources* for facts in the protocol
   - Finite process
2. **Constraint solving**
   - Backwards reachability analysis, searching for traces
   - Constraint solving with formula and graph constraints
   - Build a dependency graph to represent protocol executions
   - Solved forms have a solution corresponding to an attack trace
   - May not terminate

Tamarin's constraint solving algorithm

function $\textsc{Solve}(P \vDash_E \varphi)$

  $\hat{\varphi} \leftarrow \neg\varphi$ rewritten into negation normal form

  $\Omega \leftarrow \{\{\hat{\varphi}\}\}$

  while $\Omega \neq \emptyset$ and $\mathit{solved}(\Omega) = \emptyset$ **do**

    choose $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_k\}$ such that $\Gamma \in \Omega$

    $\Omega \leftarrow (\Omega \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}$

  if $\mathit{solved}(\Omega) \neq \emptyset$ **then**

    return "attack(s) found: ", $\mathit{solved}(\Omega)$

  else

    return "verification successful"

Constraint reduction

- A **constraint reduction** rule transforms a constraint system into a set of constraints systems

$$\Gamma \rightsquigarrow \{\Gamma_1, \ldots, \Gamma_k\}$$

- The relation is defined by a set of **reduction rules**
  - Logical rules work on formula constraints
  - Graph rules work on node and edge constraints
- Every constraint reduction rule is *sound* and *complete*, i.e., it preservers the set of solutions
- However, the problem is **undecidable**; we cannot guarantee termination!

(Some) Constraint solving rules

Trace formula reduction

$\mathbf{S}_{\approx}$:	$\Gamma \rightsquigarrow_P \|_{\sigma \in unify_{AC}(t_1,t_2)} (\Gamma\sigma)$	if $(t_1 \approx t_2) \in \Gamma$ and $t_1 \neq_{AC} t_2$
$\mathbf{S}_{\doteq}$:	$\Gamma \rightsquigarrow_P \Gamma\{i/j\}$	if $(i \doteq j) \in \Gamma$ and $i \neq j$
$\mathbf{S}_{@}$:	$\Gamma \rightsquigarrow_P \|_{ri \in [P]^{DH} \cup \{\mathsf{ISEND}\}} \|_{f' \in acts(ri)} (i : ri, f \approx f', \Gamma)$	if $(f@i) \in \Gamma$ and $(f@i) \notin_{AC} as(\Gamma)$
$\mathbf{S}_{\perp}$:	$\Gamma \rightsquigarrow_P \perp$	if $\perp \in \Gamma$
$\mathbf{S}_{\neg,\approx}$:	$\Gamma \rightsquigarrow_P \perp$	if $\neg(t \approx t) \in_{AC} \Gamma$
$\mathbf{S}_{\neg,\doteq}$:	$\Gamma \rightsquigarrow_P \perp$	if $\neg(i \doteq i) \in \Gamma$
$\mathbf{S}_{\neg,@}$:	$\Gamma \rightsquigarrow_P \perp$	if $\neg(f@i) \in \Gamma$ and $(f@i) \in as(\Gamma)$
$\mathbf{S}_{\neg,\lessdot}$:	$\Gamma \rightsquigarrow_P (i \lessdot j, \Gamma) \| (\Gamma\{i/j\})$	if $\neg(j \lessdot i) \in \Gamma$ and neither $i \lessdot_\Gamma j$ nor $i = j$
$\mathbf{S}_{\vee}$:	$\Gamma \rightsquigarrow_P (\phi_1, \Gamma) \| (\phi_2, \Gamma)$	if $(\phi_1 \vee \phi_2) \in_{AC} \Gamma$ and $\{\phi_1, \phi_2\} \cap_{AC} \Gamma = \emptyset$
$\mathbf{S}_{\wedge}$:	$\Gamma \rightsquigarrow_P (\phi_1, \phi_2, \Gamma)$	if $(\phi_1 \wedge \phi_2) \in_{AC} \Gamma$ and not $\{\phi_1, \phi_2\} \subseteq_{AC} \Gamma$

Proof Methods

Proof trees

- Tamarin uses constraint solving to prove or disprove lemmas, where each constraint reduction step generates one or more new constraint systems
- This leads to a **proof tree**, which is visible in the GUI, or output when Tamarin is run on the command line
- There can be any number of “cases” (including zero), which must be resolved
- The **qed** symbol marks the end of a list of cases

Proof methods

```
lemma trace:
  exists-trace
  "∃ a b #i. Action(a,b) @ #i"
simplify
solve( Fact ( t1, t2 ) ▶ #i1 )
  case Fact_1
  solve( !KU( t1 ) @ #vk )
    case Fact_2
    solve( (#i < #j) ‖ (#j < #i) )
      case case_1
      solve( splitEqs(i) )
        case r_1
        solve( (#vl,0) ~~> (#vk,0) )
          case r_2
          by sorry
        next
          case r_3
          by sorry
      qed
    qed
  qed
qed
```

sorry

Special "proof method" that proves nothing. Used as a placeholder.

TAMARIN gives us several options to replace `sorry` with an actual proof:

1. **simplify**
2. **induction**

a. **autoprove**
b. **autoprove** proof-depth bound 5
s. **autoprove** for all lemmas

Proof methods

```
lemma trace:
  exists-trace
  "∃ a b #i. Action(a,b) @ #i"
simplify
solve( Fact ( t1, t2 ) ▸ #i1 )
  case Fact_1
  solve( !KU( t1 ) @ #vk )
    case Fact_2
    solve( (#i < #j) ‖ (#j < #i) )
      case case_1
      solve( splitEqs(i) )
        case r_1
        solve( (#vl,0) ~~> (#vk,0) )
          case r_2
          by sorry
        next
          case r_3
          by sorry
      qed
    qed
  qed
qed
```

simplify

Translate a formula's negation into constraints. Typically the first step.

induction

Prove a lemma using induction on the length of the trace. Only possible as the first proof step.

Proof methods

```
1  lemma trace:
2    exists-trace
3    "∃ a b #i. Action(a,b) @ #i"
4  simplify
5  solve( Fact ( t1, t2 ) ▶ #i1 )
6    case Fact_1
7    solve( !KU( t1 ) @ #vk )
8      case Fact_2
9      solve( (#i < #j) ∥ (#j < #i) )
10       case case_1
11       solve( splitEqs(i) )
12         case r_1
13         solve( (#vl,0) ~~> (#vk,0) )
14           case r_2
15           by sorry
16         next
17           case r_3
18           by sorry
19       qed
20     qed
21   qed
22 qed
```

Premise constraints (line 5)

Find the origin of facts from protocol rules.

Action constraints (line 7)

Solve formula constraints, such as action fact requirements or intruder detection constraints.

Disjunction (line 9)

Turn a disjunction inside a formula into a case distinction at the constraint system level.

Proof methods

```
1  lemma trace:
2    exists-trace
3    "∃ a b #i. Action(a,b) @ #i"
4  simplify
5  solve( Fact ( t1, t2 ) ▶ #i1 )
6    case Fact_1
7    solve( !KU( t1 ) @ #vk )
8      case Fact_2
9      solve( (#i < #j) ∥ (#j < #i) )
10       case case_1
11       solve( splitEqs(i) )
12         case r_1
13         solve( (#vl,0) ~~> (#vk,0) )
14           case r_2
15           by sorry
16         next
17           case r_3
18           by sorry
19       qed
20     qed
21   qed
22 qed
```

Equation split	**(line 11)**
Perform a case split on different possible substitutions.	

Deconstruction chain	**(line 13)**
Compute whether the adversary can extract a given term from some message.	

Proof methods

```
lemma trace:
  exists-trace
  "∃ a b #i. Action(a,b) @ #i"
simplify
solve( Fact ( t1, t2 ) ▶ #i1 )
  case Fact_1
  solve( !KU( t1 ) @ #vk )
    case Fact_2
    solve( (#i < #j) ‖ (#j < #i) )
      case case_1
      solve( splitEqs(i) )
        case r_1
        solve( (#vl,0) ~~> (#vk,0) )
          case r_2
          by contradiction /* cyclic */
        next
          case r_3
          by sorry
        qed
      qed
    qed
  qed
qed
```

contradiction

TAMARIN has found a contradiction to the current constraint system. For example, circular dependencies or formulas evaluating to false. This means that there is no solution for the current constraint system.

Proof methods

```
lemma trace:
  exists-trace
  "∃ a b #i. Action(a,b) @ #i"
simplify
solve( Fact ( t1, t2 ) ▸ #i1 )
  case Fact_1
  solve( !KU( t1 ) @ #vk )
    case Fact_2
    solve( (#i < #j) ‖ (#j < #i) )
      case case_1
      solve( splitEqs(i) )
        case r_1
        solve( (#vl,0) ~~> (#vk,0) )
          case r_2
          by contradiction /* cyclic */
        next
          case r_3
          SOLVED // trace found
      qed
    qed
  qed
qed
```

SOLVED

TAMARIN has solved the constraint system; no more proof methods are applicable. Typically means that we have found an attack.

Proof method annotations

The list of currently available proof methods can have annotations:

- An action constraint is **currently deducible** when it is composed only from public constants and does not contain private function symbols, or when it can be extracted from a sent message using only unpairing or inversion
- An action constraint is **probably constructible** when it concerns a message that does not contain a fresh name or a fresh variable, and therefore can likely be constructed by the adversary
- An action constraint is **useful** when it appears in specific ways in the formulas of the constraint system

Avoiding loops

- To avoid loops when solving premise constraints, Tamarin computes a set of premises, called **loop breakers**
- Idea: Consider a graph containing a node for each rule, and an edge between two rules, if the second one has a premise fact that is part of the conclusion facts of the first one
- This graph over-approximates possible sequences of rules; any potential looping sequence of rule instances will show up as a **cycle**
- The goal is then to **remove a minimal set of premises** (the loop breakers) so that the remaining graph has no cycles
- Not a unique set; Tamarin might not find the “optimal” solution

Heuristics

- Tamarin uses **heuristics** to decide which proof method to apply
- These play an important role in whether Tamarin terminates and, if it does, how quickly (i.e., its efficiency)
- Heuristics have no influence on the result's correctness; **any** conclusion obtained by Tamarin is **always correct**
- The default options is the *smart* heuristic
    - Works well on many examples
    - Prioritizes chain constraints, disjunctions, premise constraints, action constraints, and adversary knowledge that includes private or fresh terms (in this order)
    - *Probably constructible* and *currently deducible* constraints are assigned lower priority and loop breakers are delayed

Summary

Next lecture

The Tamarin prover

- Introduction to the Tamarin prover
- **Office hours** to help with installation on Friday, December 13th

Before that: **midterm exam (2.12. at 14:15)**

- Passing the exam is a requirement to start the project
- The exam will cover topics from lectures 0-5
- Bring a **pen** and your **student id**
- This is a closed-book exam; no notes allowed!

Reading material

Recommended reading:
[Bas+24, Ch. 7.4-7.8], [Mei13, Ch. 8.3-8.4], [Sch+12]

[Bas+24] D. Basin, C. Cremers, J. Dreier, and R. Sasse. **Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide.** Draft v0.5. Sept. 2024.

[Mei13] S. Meier. **Advancing Automated Security Protocol Verification.** PhD thesis. ETH Zurich, 2013.

Reading material

[Sch+12] B. Schmidt, S. Meier, C. Cremers, and D. Basin. **Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties.** In: 2012 IEEE 25th Computer Security Foundations Symposium. 2012.

Further reading

Optional reading: [CD05], [EMS08]

[CD05] H. Comon-Lundh and S. Delaune. **The Finite Variant Property: How to Get Rid of Some Algebraic Properties.** In: Proceedings of the 16th International Conference on Term Rewriting and Applications. 2005.

[EMS08] S. Escobar, J. Meseguer, and R. Sasse. **Effectively Checking the Finite Variant Property.** In: Rewriting Techniques and Applications. 2008.