Università degli Studi di Trento

Testing Decision Procedures for Security-by-Contract

Nataliia Bielova, Ida Siahaan
University of Trento

Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security
FCS-ARSPA-WITS'08
Carnegie Mellon University, Pittsburgh – USA
June 21-22, 2008

Outline

- Motivation
- Security-by-Contract (SxC)
  - Concepts
  - Workflow
- Contract/Policy Matching
  - Specifications language
  - Automata Modulo Theory (AMT)
  - On-the-Fly Model Checking with Decision Procedure
- Prototype Implementation and Experiments
- Conclusions
  - Issues yet to be addressed

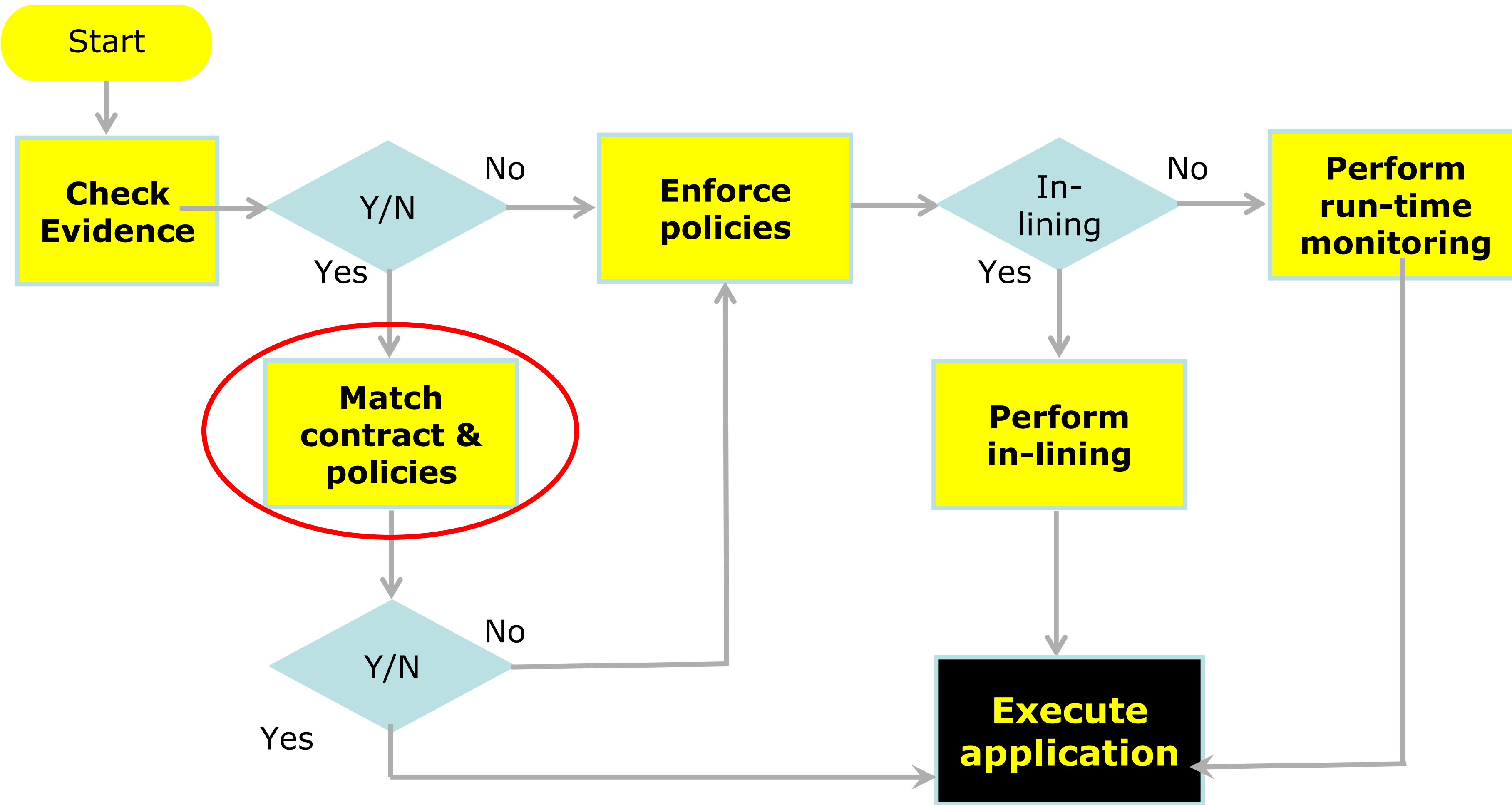
Motivations

- A validation infrastructure exists
- Mobile devices are increasingly popular and powerful
- Lack of applications for mobile devices
  - A signature is checked on the device
  - No semantics is attached to it
- Some technologies exist
  - Static analysis to prove program properties
  (Leroy et al., Morriset et al., Fournet et al.)
  - Monitor generation for complex properties
  (Havelund & Rosu, Erlingsson et al., Hamlen et al., Ligatti et al.)

Security-by-Contract (SxC) Key Concepts

- Key idea: (Dragoni et al. EuroPKI'07)
  - the digital signature should not just certify the origin of the code but rather bind together the code with a contract
  - Model-Carrying Code – model that captures the security-relevant behavior of code
  - Design-by-contract
- Contract carried by application;
  - Claimed Security behavior of application
  - (Security) interactions with its host platform
  - Maybe with Proof that code satisfies contract
- Policy specified by a platform.
  - Desired Security behavior of application
  - Fine-grained resource control
- End Users' Distilled Security Requirements e.g:
  - NETwork connectivity
  - PRIvate information management
  - INTeraction with other applets
  - Power consumption

SxC Workflow – User's View



Contributions

- Algorithms:
  - meta-level algorithm (Dragoni et al. EuroPKI'07)
  - mathematical structure for algorithm to do the matching (Massacci & Siahaan, NordSec'07)
- Does it work in practice?
  - contract/policy matching implementation (Dragoni et al. ARES'08)
- Our main contributions of this paper:
  - integration issues with decision procedure solver NuSMV integrated with its MathSAT libraries
  - performance analysis of the integration design alternatives:
    - construction of expressions
    - initialization of solver
    - caching of temporary results

Language of contract/policy

- ConSpec – automata-based language
- The specifications in ConSpec is suitable for all phases of Security-By-Contract lifecycle
  - Contract / Policy Matching
  - Monitor In-lining
- Contract and Policy are mapped to the specific automata representation
- Matching = Language inclusion
  - Actions allowed by the contract subset actions allowed by the policy

Contract vs Policy

Contract = What you Claim

Rules

- Used Methods
- Bounds on Methods Args
- Bounds on ret Values
- Allowed Sequences
- *Achievable Obligations*

Policy = What you should at most do

Rules

- Possible Methods
- Constraints on Methods Args
- Constraints on ret Values
- History-based access control
- *Desired Obligations*

Language Inclusion = Simulation of Finite Automata ?

(Massacci & Siahaan, NordSec'07) (Massacci & Siahaan, PLAS'08)

(Sekar et al.)

What kind of automaton?

- We need "infinite" edges to describe policies

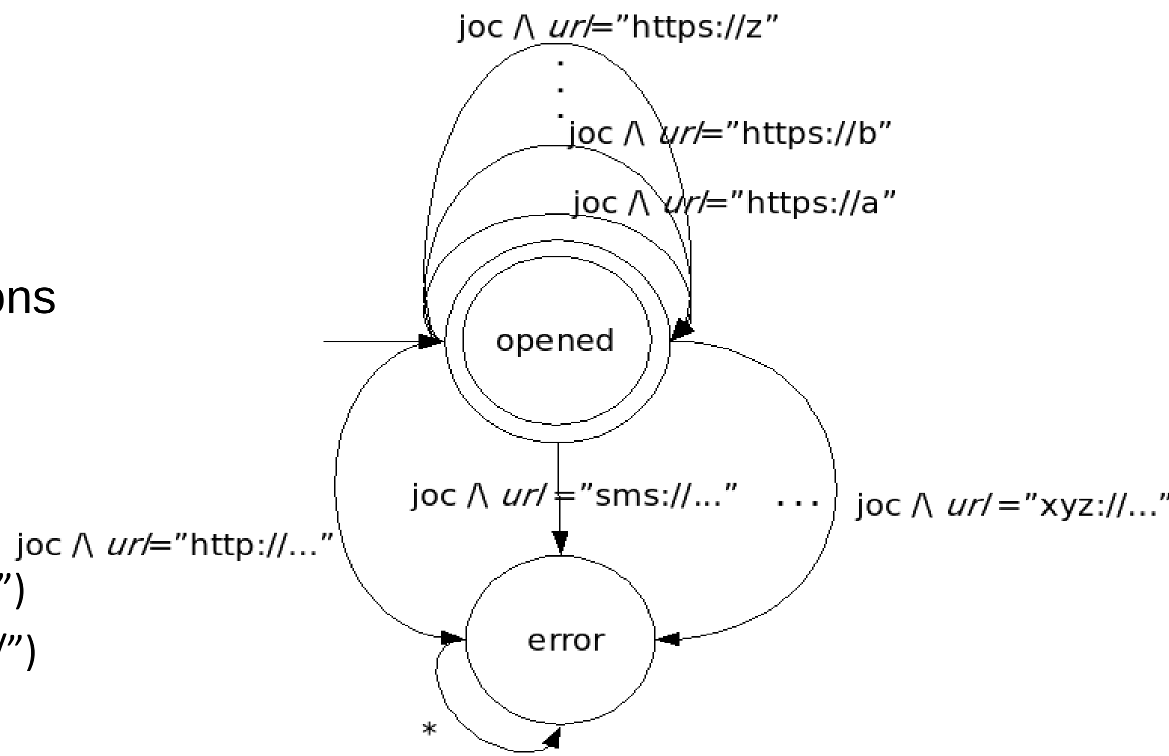
CONTRACT:
The application only uses HTTPS network connections

Abbreviations for JAVA APIs:

joc = io.Connector.open(url)

p(url) = url.startsWith(”http://”)

s(url) = url.startsWith(”https://”)



Automata Modulo Theory ($\mathcal{AMT}$)

- AMT
  - Finite state automata with “infinite” edges
  - BUT Finitely represented with Expressions:

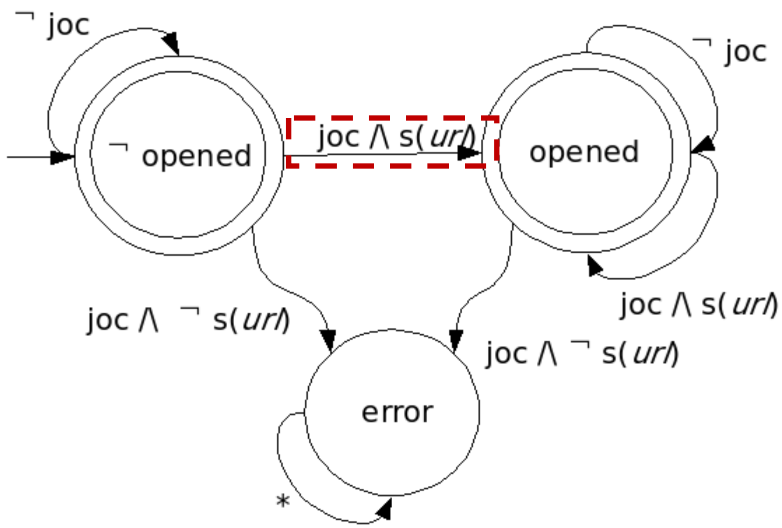
    p = io.Connector.open(url) &&
    (url.startsWith(”http://”) || url.startsWith(”https://”))
- Matching = Language inclusion can be reduced to an emptiness test:

  $L_{AutC} \square L_{AutP} \square L_{AutC} \cap L_{NEG\ AutP} = \varnothing$
- *Search for counterexamples:*
  - Path allowed by contract but NOT allowed by policy

Contract vs Policy in $\mathcal{AMT}$

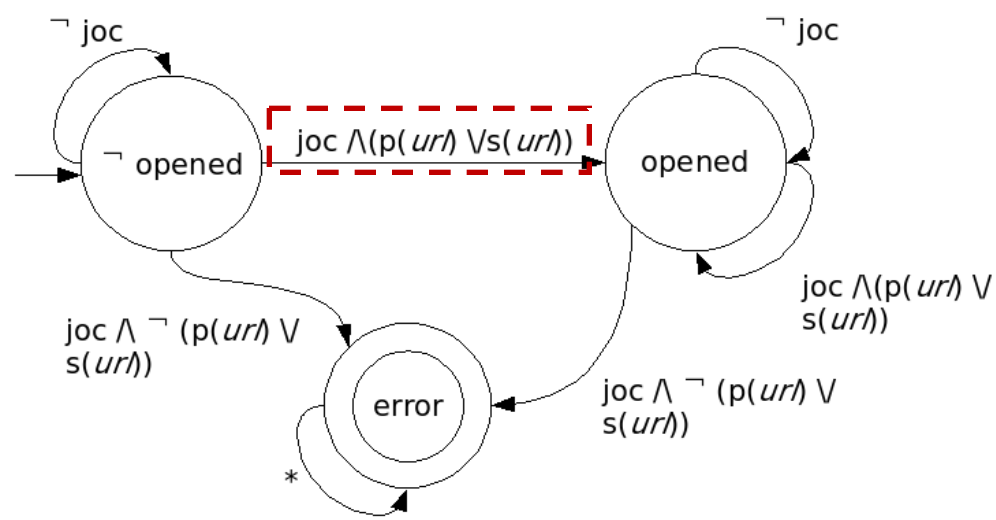
CONTRACT:
The application only uses HTTPS network connections

Automaton:



POLICY:
The application uses only high-level (HTTP, HTTPS) network connections

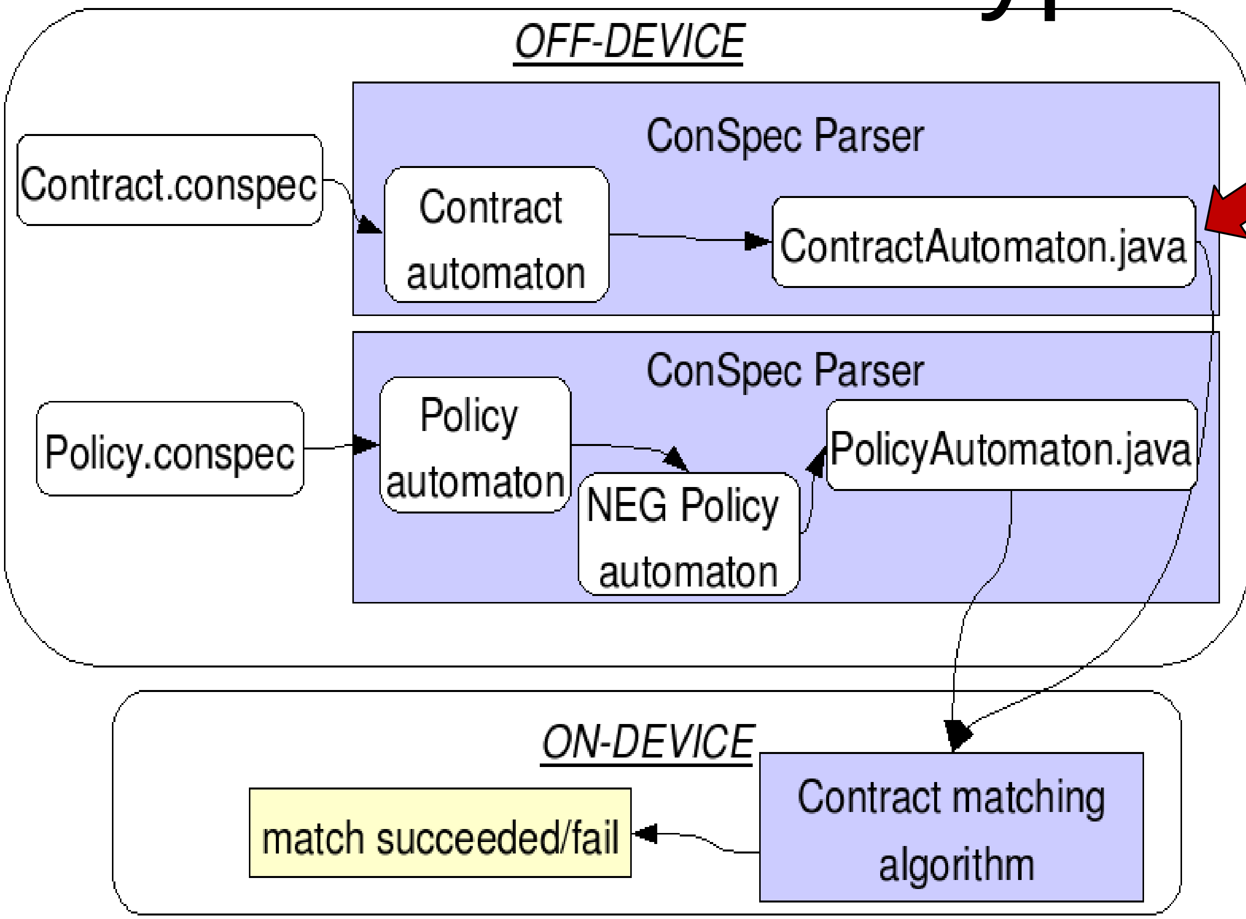
Negated automaton:



Abbreviations for JAVA APIs:

joc = io.Connector.open(url)
p(url) = url.startsWith(”http://”)
s(url) = url.startsWith(”https://”)

Architecture of Matching Prototype




Java source file

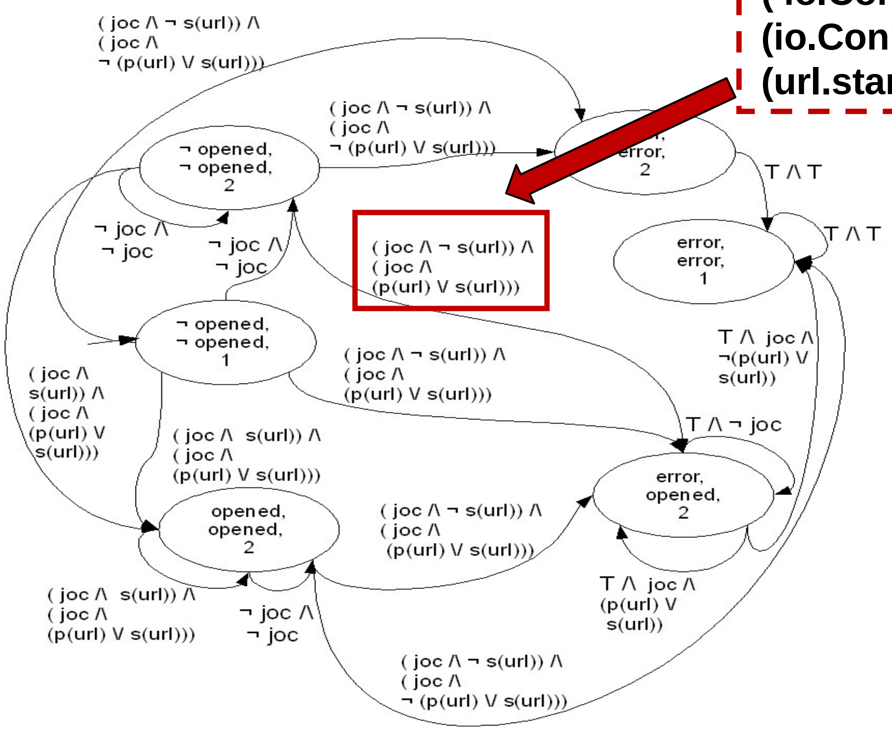
```
import org.s3ms.copv.automaton.*;
import org.s3ms.copv.expressions.*;
import org.s3ms.copv.matcher.*;

pub   c class PolicyRules{
      PolicyRules(){

public static AutomatonMTT Create(){
//========******* AUTOMATA FOR Policy *******=======
//--------- Initialization of the states --------
HashSet<State> states_1 = new HashSet<State>(2,1);
//--------- State errorState_1 initialization --------
HashMap<String, Object> fieldserrorState_1_1 =
ew HashMap<String, Object>(0,1);
//--------- Add the fields to the state --------
State errorState_1 = new State(fieldserrorState_1_1);
states_1.add(errorState_1);
// =================******* FROM STATE ERRORSTATE_1*******==
HashSet<Transition> from_errorState_1 =
new HashSet<Transition>(1,1);
SpecificBoolExp e_0_1 =  new SpecificBoolExp(false);
from_errorState_1.add(new Transition(e_0_1, errorState_1));
transitionsMap_1.put(errorState_1,
new StateDef(false, true, from_errorState_1));
```

Off-line: mapping to automaton – expensive operation complementation – for optimization
On-line: On-The-Fly algorithm

On-the-Fly Model Checking

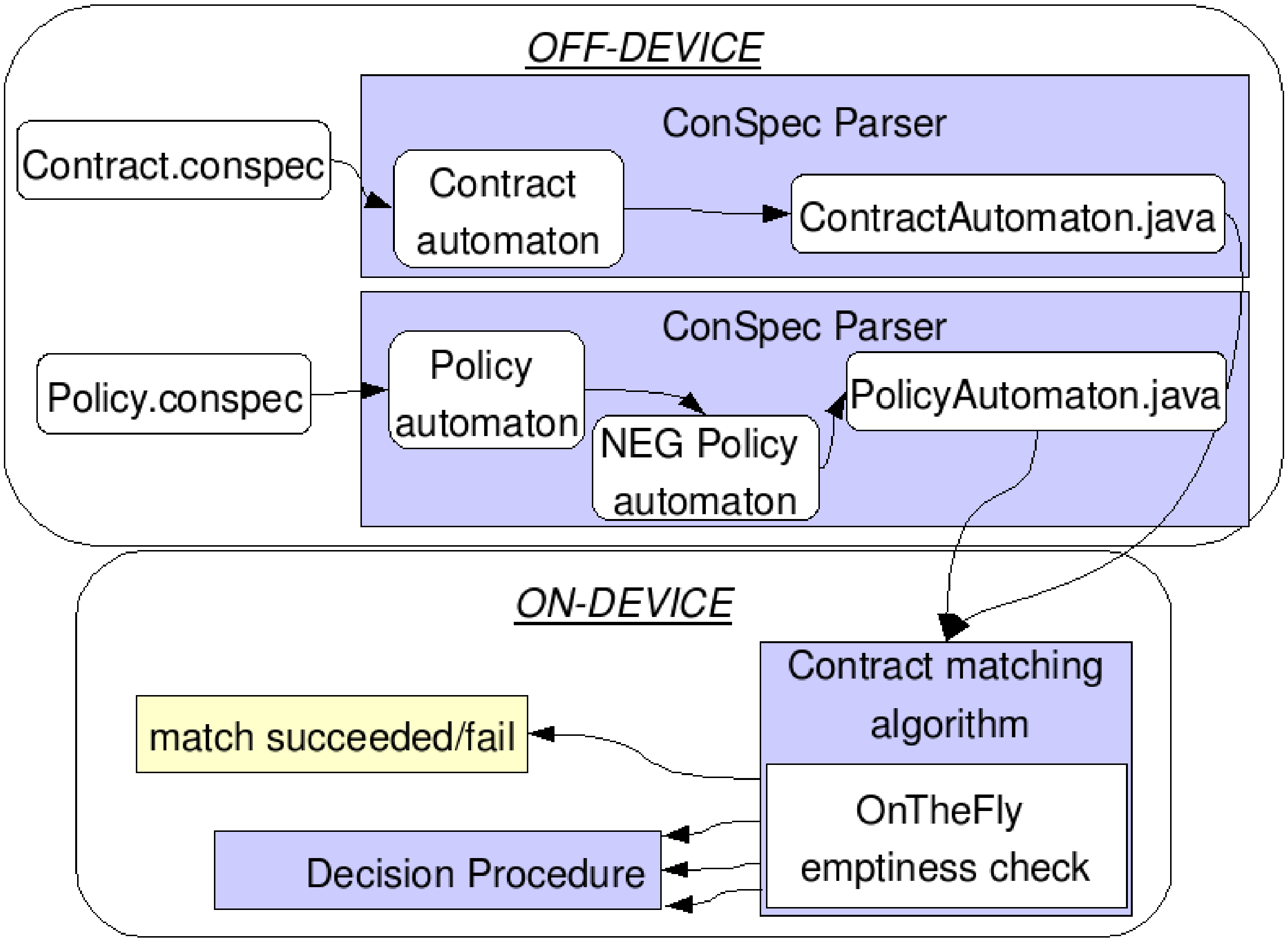
- The search space for counterexample (a trace that satisfies the Contract and violates the Policy)

(io.Connector.open(url) /\ !url.startsWith("https://"))/
(io.Connector.open(url) /
(url.startsWith("http://") \/ url.startsWith("https://")))



On-the-Fly Algorithm with decision procedure for SMT
Interaction with the solver of math expression NuSMV
for satisfiability checks.

On-the-Fly Model Checking with Decision Procedure



Design Decisions

- **One vs Many**
  - only one instance of solver or a new instance of the solver every call of decision procedure
- **MUTEX SOLVER**
  - all the method names are declared as mutex constants at the moment of declaring all variables
  - expression sent to the solver: method = name^cond^otherConds
- **MUTEX MC**
  - allows the on-the-fly algorithm to check whether method names are the same
  - expression sent to the solver if check passed: cond ^ otherConds
- **PRIORITY MC**
  - guards are evaluated using priority OR
  - expressions as lemmas: cond
- **CACHING MC**
  - many edges will be traversed again and again => caching the results of the matching
  - Solver has a caching mechanism that could be equally used: CACHING SOLVER

Experiments on Desktop and on Device

- Implemented on a Java platform for a Desktop PC
  - Intel(R) Pentium(R) D CPU 3.40GHz,3389.442MHz, 1.99GB of RAM, 2048 KB cache size) with operating system Linux version 2.6.20-16-generic, Kubuntu 7.04 (Feisty Fawn)
- Some experimental results on .NET implementation for a Mobile platform i.e. ported to HTC P3600
  - 3G PDA phone with ROM 128MB, RAM 64MB, SamsungR SC32442A processor 400MHz

Problem	Contract	Policy	SC	TC	SP	TP
P1	size_100_512_contract.pol	size_10_1024_policy.pol	2	4	2	4
P2	maxKB512_contract.pol	maxKB1024_policy.pol	2	4	2	4
P3	noPushRegistry_contract.pol	oneConnRegistry_policy.pol	2	3	3	9
P4	notCreateRS_contract.pol	notCreateSharedRS_policy.pol	2	4	2	4
P5	pimNoConn_contract.pol	pimSecConn_policy.pol	3	7	3	9
P6	2hard_contract.pol	2hard_policy.pol	3	7	3	7
P7	httpLcontract.pol	httpsL_policy.pol	3	7	3	7
P8	3hard_contract.pol	3hard_policy.pol	3	7	3	7
P100	noSMS_contract.pol	100SMS_policy.pol	2	4	102	304

Problems Suit

Results on Desktop and on Device

(a) Running Problem Suit

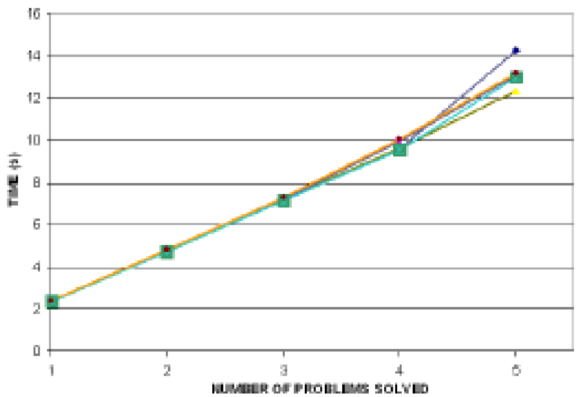
MUTEX_MC ONE_INSTANCE CACHING_SOLVER									
Problem	**Desktop**				**Mobile**				**Result**
	ART (s)	**CRT (s)**	**SV**	**TV**	**ART (s)**	**CRT (s)**	**SV**	**TV**	
P1	2.4	2.4	2	6	4.3	4.3	2	6	Match
P2	2.4	4.8	2	6	4.1	8.4	2	6	Match
P3	2.4	7.2	3	11	3.9	12.3	3	11	Match
P4	2.4	9.6	2	6	4.0	16.3	2	6	Match
P5	4.7	14.3	3	11	4.1	20.4	3	11	Match
P6	2.9	2.9	4	4	3.8	3.8	3	6	Not Match
P7	2.8	5.7	5	7	3.8	7.6	2	4	Not Match
P8	2.9	8.6	5	7	3.8	11.4	3	6	Not Match
P100	9.3	9.3	102	307	11.3	11.3	102	307	Match

(b) Abbreviations

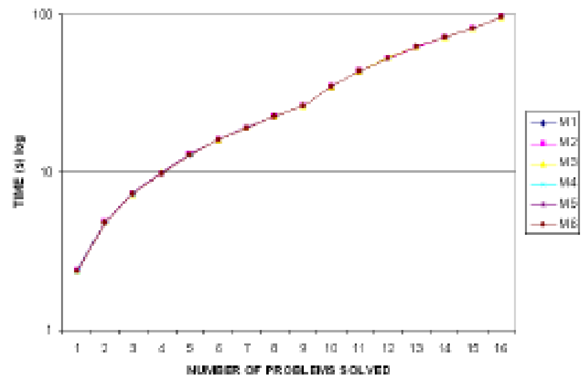
ART: Average Runtime for 10 runs SV: Number of Visited States
CRT: Cumulative Average Runtime TV: Number of Visited Transitions

Running Problem Suit 10 Times

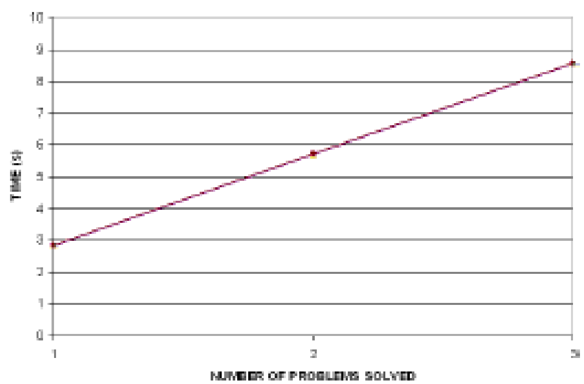
Performance Analysis of Integration Design Alternatives



(a) Match succeeds for real policies



(b) Matches among synthetic contracts and policies



(c) Match fails for real policies

M1: MUTEX_MC ONE_INSTANCE CACHING_SOLVER
M2: MUTEX_SOLVER ONE_INSTANCE CACHING_SOLVER
M3: PRIORITY_MC ONE_INSTANCE CACHING_SOLVER
M4: MUTEX_MC ONE_INSTANCE CACHING_MC
M5: MUTEX_SOLVER ONE_INSTANCE CACHING_MC
M6: PRIORITY_MC ONE_INSTANCE CACHING_MC

(d) Abbreviations for Configurations

Cumulative response time of matching algorithm on Desktop PC

Issues yet to be addressed

- Encoding of history dependent policies
  - allow certain strings we saw in the past
    - Eg connect only to url in the JAR manifest
    - Combine AMT with History Dependent Automata (Montanari & Pistore 1998)
    - Combine AMT with Extended Finite State Automata (Sekar et al. 2002)
- Infinite expressions
  - allow concrete run of infinite domains
    - Eg natural number not limited to some maximum length
    - Combine AMT with Finite Memory Automata (Kaminski & Francez 1994)

Conclusions

- Security-by-Contract
  - Always consider complete lifecycle monitoring is the end
  - Matching: be able to check that claimed security behavior of what you want to run is good for your security policy
- Automata Modulo Theory
  - Invented for security policies of mobile code but…
  - usable for any security policy with a finite control structure but potentially infinite data
    - (secure workflows, protocol analysis, control-flow analysis etc.)
  - IF polynomial theory for deciding edges THEN Practical
- Implementation of Contract/Policy Matching
  - Current implementation uses PRIORITY MC ONE INSTANCE CACHING MC configuration.
    - PRIORITY MC: the nature of rules in policies i.e priority OR
    - MUTEX SOLVER does not allow empty methods e.g. $\neg m_i \wedge \neg m_j$ i.e possible in the matching algorithm
    - ONE INSTANCE: garbage collection problem
    - CACHING MC: save calls to solver for the already solved rules
- Experiments on Desktop and on Device