Imperial College London

Numerical Aggregation of Trust Evidence: From Foundations to Reasoning Support

Michael Huth and Jim Huan-Pu Kuo
Imperial College London
in part joint work with
Jason Crampton and Charles Morisset

19 November 2013

1

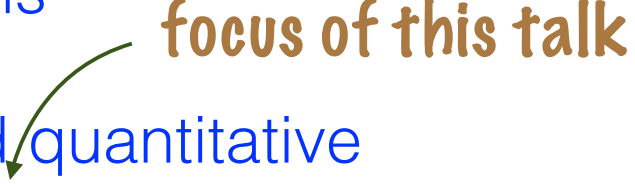
Outline of presentation

- Motivation
- Peal: **p**luggable **e**vidence **a**ggregation **l**anguage
- Generating verification conditions for Peal
- Experimental results
- Future work and conclusions

2

Our Intel project aim

- understand how software annotations can generate trust evidence
- show that such evidence leads to policies that can effectively guard rail executions
- do this for both qualitative and quantitative evidence and develop verification support for this
- study programmers' intent as one source of evidence



3

Trust evidence

- increasingly, *evidence for trust is numeric*
- e.g. *reputations* of web sites
- e.g. *age* of software

- e.g. *statistical information* about past behavior of subjects
- e.g. *probability* of one machine connecting to another

Example

- numeric evidence may be needed even in seemingly qualitative settings
- e.g. you just downloaded the MacTex 2012 package and it does not have a matching hash
- *how likely is it that this downloaded software is maliciously corrupted?*
- conditional on which browser you use (e.g. Chrome does things to your file!)

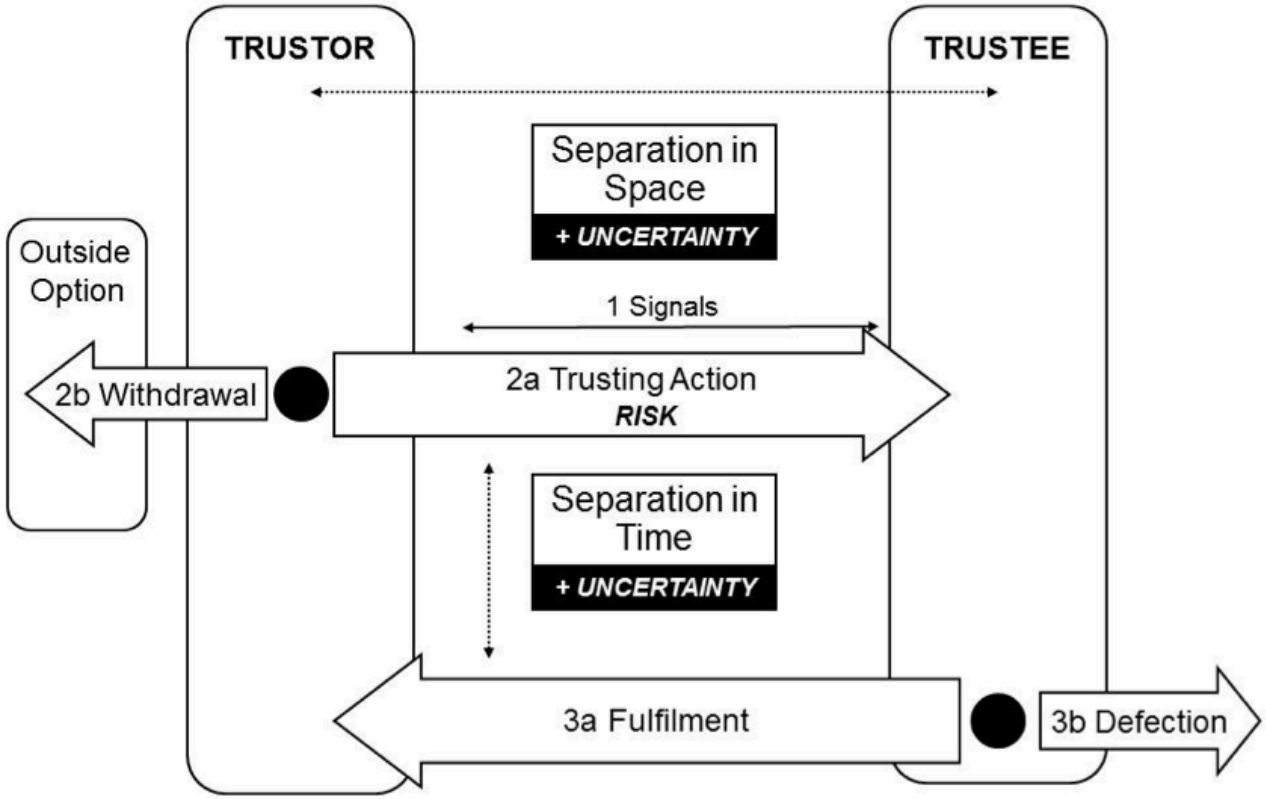
5

Numeric aggregation and its verification

- want to use numeric/non-numeric evidence to produce recommendations or obligations
- which then should inform access-control decisions
- and at all sorts of hardware and software interfaces (e.g. forgetful loaders, social networks, infrastructure integrity, risk postures of organizations)
- how can we verify such aggregation and what does verification mean here?

6

Trust-mediated Interactions

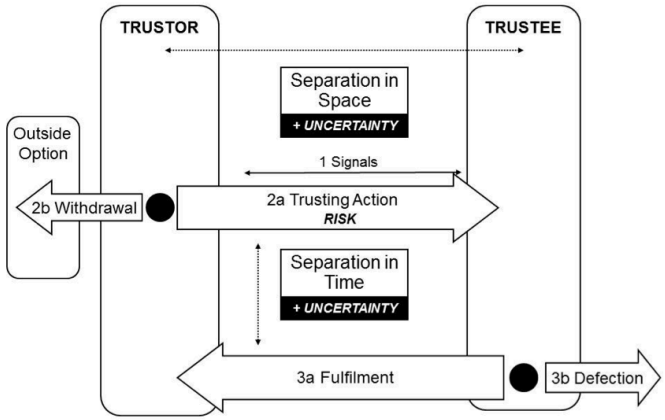


Jens Riegelsberger, Martina Angela Sasse, and John D. McCarthy. The mechanics of trust: A framework for research and design. Int. J. Hum.-Comput. Stud., 62(3):381–422, 2005.

7

Trust & Assurance

- Economic incentive to use trust: assurance techniques are expensive
- But trust signals less reliable in digital domain
- *Want to use such signals and combine them with assurance techniques*
- ... and in a tuneable manner



8

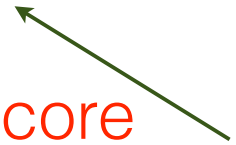
Outline of presentation

- Motivation
- Peal: pluggable evidence aggregation language
- Generating verification conditions for Peal
- Experimental results
- Future work and conclusions

9

Policies as blocks of aggregated trust signals

op ::= + | min | max | * | ...
rule ::= if (q) score
pol ::= op(rule*) default score



scope for extensions

- policy (*pol*) returns a score:
- first, collect all scores of rules in policy whose predicate (i.e. *trust or distrust signal q*) is true
- second, apply *op* to all these scores to get result; if no *q* is true, return default score

10

Full Peal syntax

op ::= + | * | min | max | ...
rule ::= if (q) score
pol ::= op(rule*) default score
pSet ::= pol | max(pSet,pSet) |
min(pSet,pSet)
cond ::= th < pSet | pSet <= th

- *op* aggregates scores of true predicates in policy
- policy isA policy set, combine policy sets (*pSet*) with *min* or *max*
- *cond*: compares values of *pSet* with thresholds

11

Example

- Alice wants to pay Bob via PayPal on Facebook

$$b_1 = +((if\ (lowCostTransaction\ 0.3)\ (if\ enoughMutualFriends\ 0.1)\ (if\ enoughMutualFriendsNormalized\ 0.2))\ default\ 0$$

$$b_2 = min((if\ (highCostTransaction\ 0.1)\ (if\ aFriendOfAliceUnfriendedBob\ 0.2)\ (if\ aFriendOfAliceVouchesForBob\ 0.6))\ default\ 1$$

$$cond = 0.5 < min(b_1, b_2)$$

Analysis example:

Does the behavior of cond *change when threshold 0.5 changes to 0.6?*

Definition of signals

$$
\begin{aligned}
lostCostTransaction &= (amountAlicePays < 100) \\
highCostTransaction &= (1000 < amountAlicePays) \\
enoughMutualFriends &= (4 < numberOfMutualFriends) \\
enoughMutualFriendsNormalized &= (numberOfBobsFriends < \\
&\qquad 100 * numberOfMutualFriends)
\end{aligned}
$$

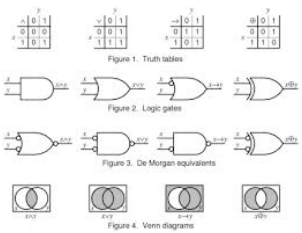
- transaction risk relies on amount that Alice means to pay, amount of 500 does not signal anything!
- other signals also rely on state of relationship graph, e.g. the number of mutual friends
- Last predicate protects against Bob having way too many friends (e.g. celebrity)

Outline of presentation

- Motivation
- Peal: pluggable evidence aggregation language
- Generating verification conditions for Peal
- Experimental results
- Future work and conclusions

14

Logical synthesis



- For each *cond* ("*th* < *pSet*" or "*pSet* <= *th*"): synthesize propositional formula **Φ***[cond]* over *q*
- Models = determine truth values of predicates *q*
- **Φ***[cond]* is true in model iff *cond* holds in model
- Synthesized **Φ***[cond]* is basis for verification tasks
- predicates *q* themselves are/representable as formulas of first-order logic

15

Verification tasks

- Is $\Phi[cond]$ always true/false? Vacuity checking
- Do $\Phi[th < pSet]$ and $\Phi[th' < pSet]$ differ? Sensitivity analysis

- Is *cond* persistent, i.e. incomplete requests will grant access only if complete one would?

These and more complex verification tasks reduce to satisfiability checks of $\Phi[cond]$ in logic over predicates *q*

Analyze instrumentations of Φ*[cond]*

- Is Φ*[cond]* always false? Satisfiability checking of that formula, then negate result
- Is Φ*[cond]* always true? Satisfiability checking of negation of that formula, then negate result
- Do Φ*[th < pSet]* and Φ*[th' < pSet*] differ, when *th < th'*? Satisfiability checking of Φ*[th < pSet] && !*Φ*[th' < pSet]*

Etc. But satisfiability checks have to recognize logical dependencies of predicates *q* within *pSet*

17

SMT solvers

- SMT solvers: combine SAT solvers (“model checking”) with logical reasoning over theories (theorem proving)
- we have first prototype verification tool using SMT solver Z3 (Microsoft Research)
- below we will explore tradeoff between explicit synthesis of $\Phi[cond]$ (space intensive) and synthesis of symbolic version of $\Phi[cond]$
- but first we provide idea and usage of SMT solvers, using Z3 as an example

Simple Z3 code

```
(declare-const a Int)   ; example from Z3 documentation
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
```

- declares an integer, and a function that maps an (Int Bool) pair to an integer
- declares two assertions: a is greater than 10, and the value f(a,true) is less than 100
- (check-sat) directive to decide whether there is a model for conjunction of all assertions

Z3 witnesses

```
(declare-const a Int) ; example from Z3 documentation
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
(get-model)
```

- (get-model) constructs witness of satisfiability, reports exception if no such model exists

```
sat
(model
  (define-fun a () Int
    11)
  (define-fun f ((x!1 Int) (x!2 Bool)) Int
    (ite (and (= x!1 11) (= x!2 true)) 0
      0))
)
```

Scoping in Z3

```
(declare-const x Int) ; example from Z3 documentation
(declare-const y Int)
(declare-const z Int)
(push)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
(pop) ; remove the two assertions
(push)
(assert (= (+ (* 3 x) y) 10))
(assert (= (+ (* 2 x) (* 2 y)) 21))
(check-sat)
(declare-const p Bool)
(pop)
(assert p) ; error, as declaration of p was removed from the stack
```

Each analysis in our tool wrapped in a push/pop frame

Predicate definitions in Z3

```
DOMAIN_SPECIFICS
(declare-const amountAlicePays Real)
(declare-const numberOfMutualFriends Int)
(declare-const numberOfBobsFriends Int)
(assert (= lowCostTransaction (> 100.0 amountAlicePays)))
(assert (= enoughMutualFriends (< 4 numberOfMutualFriends)))
(assert (= enoughMutualFriendsNormalized
           (< numberOfBobsFriends (* 100 numberOfMutualFriends))))
(assert (= highCostTransaction (> amountAlicePays 1000.0)))
```

- Our tool for Peal verification has text zone in which such domain specifics can be added

Possible Z3 answers

- Some theories and their combination are decidable over first-order logic
- *Even if not, conjunction of all assertions may still be decided in instances*
- "sat" means that conjunction has model
- "unsat" means that conjunction has no model
- "unknown" means Z3 does not know which is the case

Explicit synthesis

- Generate verification condition $\Phi[cond]$ by translating away all references to numerics
- Captures logical nature of condition
- Output is amenable to analysis with an SMT solver
- SMT solver will *correctly reflect logical dependencies of predicates within/across policies*

Explicit synthesis for *pSet* composition

- Induction on min/max compositions
- Encodes order properties of min/max

$$\phi[min(pS_1, pS_2) \leq th] \stackrel{\text{def}}{=} \phi[pS_1 \leq th] \vee \phi[pS_2 \leq th]$$

$$\phi[max(pS_1, pS_2) \leq th] \stackrel{\text{def}}{=} \phi[pS_1 \leq th] \wedge \phi[pS_2 \leq th]$$

$$\phi[th < min(pS_1, pS_2)] \stackrel{\text{def}}{=} \phi[th < pS_1] \wedge \phi[th < pS_2]$$

$$\phi[th < max(pS_1, pS_2)] \stackrel{\text{def}}{=} \phi[th < pS_1] \vee \phi[th < pS_2]$$

x < max(a,b)
iff (x < a) or (x < b)

25

Synthesis of defaults

- pol = op((q1 s1) ... (qn sn)) default *s*
- reflect whether default is possible or not

non-default case

$$\phi[pol \leq th] \stackrel{\text{def}}{=} (\bigwedge_{i=1}^{n} \neg q_i) \vee \phi_{op}^{ndf}[pol \leq th] \qquad (s \leq th)$$

$$\phi[pol \leq th] \stackrel{\text{def}}{=} (\bigvee_{i=1}^{n} q_i) \wedge \phi_{op}^{ndf}[pol \leq th] \qquad (th < s)$$

$$\phi[th < pol] \stackrel{\text{def}}{=} (\bigwedge_{i=1}^{n} \neg q_i) \vee \phi_{op}^{ndf}[th < pol] \qquad (th < s)$$

$$\phi[th < pol] \stackrel{\text{def}}{=} (\bigvee_{i=1}^{n} q_i) \wedge \phi_{op}^{ndf}[th < pol] \qquad (s \leq th)$$

26

Synthesis of defaults

- default *s* consistent with threshold: conjunction of all negated qi as one disjunct

- otherwise, ensure at least one qi be true

$$\phi[pol \leq th] \stackrel{\text{def}}{=} (\bigwedge_{i=1}^{n} \neg q_i) \vee \phi_{op}^{ndf}[pol \leq th] \qquad (s \leq th)$$

consistent

$$\phi[pol \leq th] \stackrel{\text{def}}{=} (\bigvee_{i=1}^{n} q_i) \wedge \phi_{op}^{ndf}[pol \leq th] \qquad (th < s)$$

inconsistent

$$\phi[th < pol] \stackrel{\text{def}}{=} (\bigwedge_{i=1}^{n} \neg q_i) \vee \phi_{op}^{ndf}[th < pol] \qquad (th < s)$$

$$\phi[th < pol] \stackrel{\text{def}}{=} (\bigvee_{i=1}^{n} q_i) \wedge \phi_{op}^{ndf}[th < pol] \qquad (s \leq th)$$

27

Polarity of synthesis

- *want witnessing sets of predicates to be closed under supersets*

- *X* contained in *Y* implying *op(X) <= op(Y)* means *op* is monotone; in that case

$$\phi_{op}^{ndf}[pol \leq th] \quad \stackrel{\text{def}}{=} \quad \neg\phi_{op}^{ndf}[th < pol]$$

- *X* contained in *Y* implying *op(Y) <= op(X)* means *op* is anti-tone; in that case

$$\phi_{op}^{ndf}[th < pol] \quad \stackrel{\text{def}}{=} \quad \neg\phi_{op}^{ndf}[pol \leq th]$$

28

Monotone: *op* is *max*

- *pol = max((q1 s1) … (qn sn)) default s*
- *th < pol* if for some *qi* inequality *th < si* is true
- last slide: reduce *pol <= th* to *th < pol* case

$$\phi_{op}^{ndf}[th < pol] \stackrel{\text{def}}{=} \bigvee_{i|th<s_i} q_i$$

$$\phi_{op}^{ndf}[pol \leq th] \stackrel{\text{def}}{=} \neg\phi_{op}^{ndf}[th < pol]$$

29

Anti-tone: *op* is *min*

- *pol = min((q1 s1) ... (qn sn)) default s*
- *pol <= th* if there is some *qi* with *si <= th*
- now reduce *th < pol* to *pol <= th* case

$$\phi_{op}^{ndf}[pol \leq th] \stackrel{\text{def}}{=} \bigvee_{i|s_i \leq th} q_i$$

$$\phi_{op}^{ndf}[th < pol] \stackrel{\text{def}}{=} \neg\phi_{op}^{ndf}[pol \leq th]$$

30

Monotone: *op* is *+*

- let *pol* be *+((q1 s1) ... (qn sn)) default s*
- sort predicates *qi* in *pol* by ascending score order, e.g. the vector [0.1, 0.2,0.2,0.3,0.5]
- want to find minimal index sets {i,...,j} of predicates whose score sum is greater than 0.5 ("*0.5 < pol*"), say
- here: M1 = {{4,5}, {3,5}, {2,5}, {2,3,4}, {1,3,4}, {1,2,4}}
- explore from highest score, expand only if partial sum ti = s0 + ... + si can get current mass over threshold 0.5

31

Computing all minimal index sets: *M1*

unit of +

explore indexes less than this

call enumOne({},0,n+1,+) for n rules where ti = s0 + … + si

current index set

aggregation operator

$$
\begin{array}{l}
enumOne(X, acc, index, op)\ \{ \\
\quad if\ (th < acc)\ \{\ output\ X;\ \} \\
\quad else\ \{ \\
\quad\quad j = index - 1; \\
\quad\quad while\ ((0 < j) \wedge (th < op(acc, t_j))\ \{ \\
\quad\quad\quad enumOne(X \cup \{j\}, op(acc, s_j), j, op); \\
\quad\quad\quad j = j - 1; \}\}\}
\end{array}
$$

32

Example trace

th = 0.5, **s** = [0.1,0.2,0.2, 0.3, 0.5], **t** = [0.1,0.3,0.5,0.8,1.3]

```
enumOne({},0,6,+)
  enumOne({5},0.5,5,+)
    enumOne({5,4},0.8,4,+) --> {5,4}
    enumOne({5,3},0.7,3,+) --> {5,3}
    enumOne({5,2},0.7,2,+) --> {5,2}
    enumOne({5,1},0.6,1,+) --> {5,1}
  enumOne({4},0.3,4,+)
    enumOne({4,3},0.5,3,+)
      enumOne({4,3,2},0.7,2,+) --> {4,3,2}
      enumOne({4,3,1},0.6,1,+) --> {4,3,1}
    enumOne({4,2},0.5,2,+)
      enumOne({4,2,1},0.6,1,+) --> {4,2,1}
```


33

Non-default formula for +

- *pol = +((q1 s1) ... (qn sn)) default s*
- sort rules so that *s0 <= s1 <= ... <= sn*
- use enumerateOne to generate *M1* below

$$\phi_{op}^{ndf}[th < pol] \stackrel{\text{def}}{=} \bigvee_{X \in \mathcal{M}_1} \bigwedge_{i \in X} q_i$$

34

Anti-tone: *op* is *

- let *pol* be **((q1 s1) ... (qn sn)) default s*
- Need: scores in [0,1]; so multiplication * is anti-tone; sort scores in *pol* by descending order
- find minimal index sets {i,...,j} of *qi* whose score sum is <= th; *M2* = set of such minimal index sets
- explore from lowest score, recurse only if partial product ti can get current mass below threshold th

$$\phi_{op}^{ndf}[pol \leq th] \stackrel{\text{def}}{=} \bigvee_{X \in \mathcal{M}_2} \bigwedge_{i \in X} q_i$$

35

Computing all minimal index sets: *M2*

call enumTwo({},1,n+1,*) for n rules where unit = 1

enumTwo only reverses "th" comparisons of enumOne

$$
\begin{array}{l}
enumTwo(X, acc, index, op)\ \{ \\
\quad if\ (acc \leq th)\ \{\ output\ X;\ \} \\
\quad else\ \{ \\
\qquad j = index - 1; \\
\qquad while\ ((0 < j) \wedge (op(acc, t_j) \leq th)\ \{ \\
\qquad\quad enumTwo(X \cup \{j\}, op(acc, s_j), j, op); \\
\qquad\quad j = j - 1; \}\}\}
\end{array}
$$

36

Example: Z3 code

- policies b1, b2 composed with max, satisfiability of “cond” explored

cond = pSet <= 0.5
b1 = min ((q1 0.2) (q2 0.4) (q3 0.9)) default 1
b2 = + ((q4 0.1) (q5 0.2) (q6 0.2)) default 0
pSet = max(b1, b2)

```
(declare-const q4 Bool) ... ; declare variables/functions
(assert (and (and (or q1 q2 q3) (or q1 q2)) (or (and (not q4) (not q5) (not q6)) (not false)))) ; synthesised formula in red
(check-sat) ; directive to check for satisfiability
(get-model) ; extraction of witness if satisfiable
```


37

Parse tree of assertion

cond = pSet <= 0.5

b1 = min ((q1 0.2) (q2 0.4) (q3 0.9)) default 1

b2 = + ((q4 0.1) (q5 0.2) (q6 0.2)) default 0

pSet = max(b1, b2)

max <= 0.5

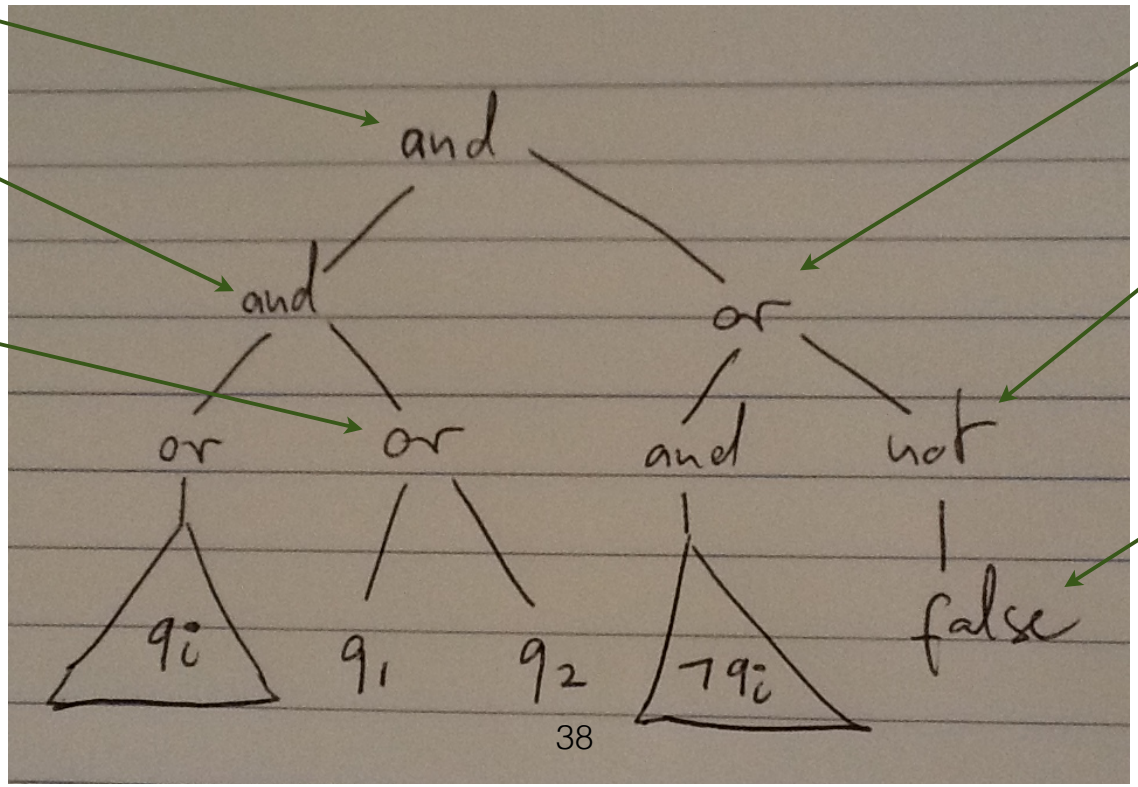
inconsistent default score

qi with si <= 0.5

consistent default score

make + monotone

M1 empty



38

Naming mechanisms

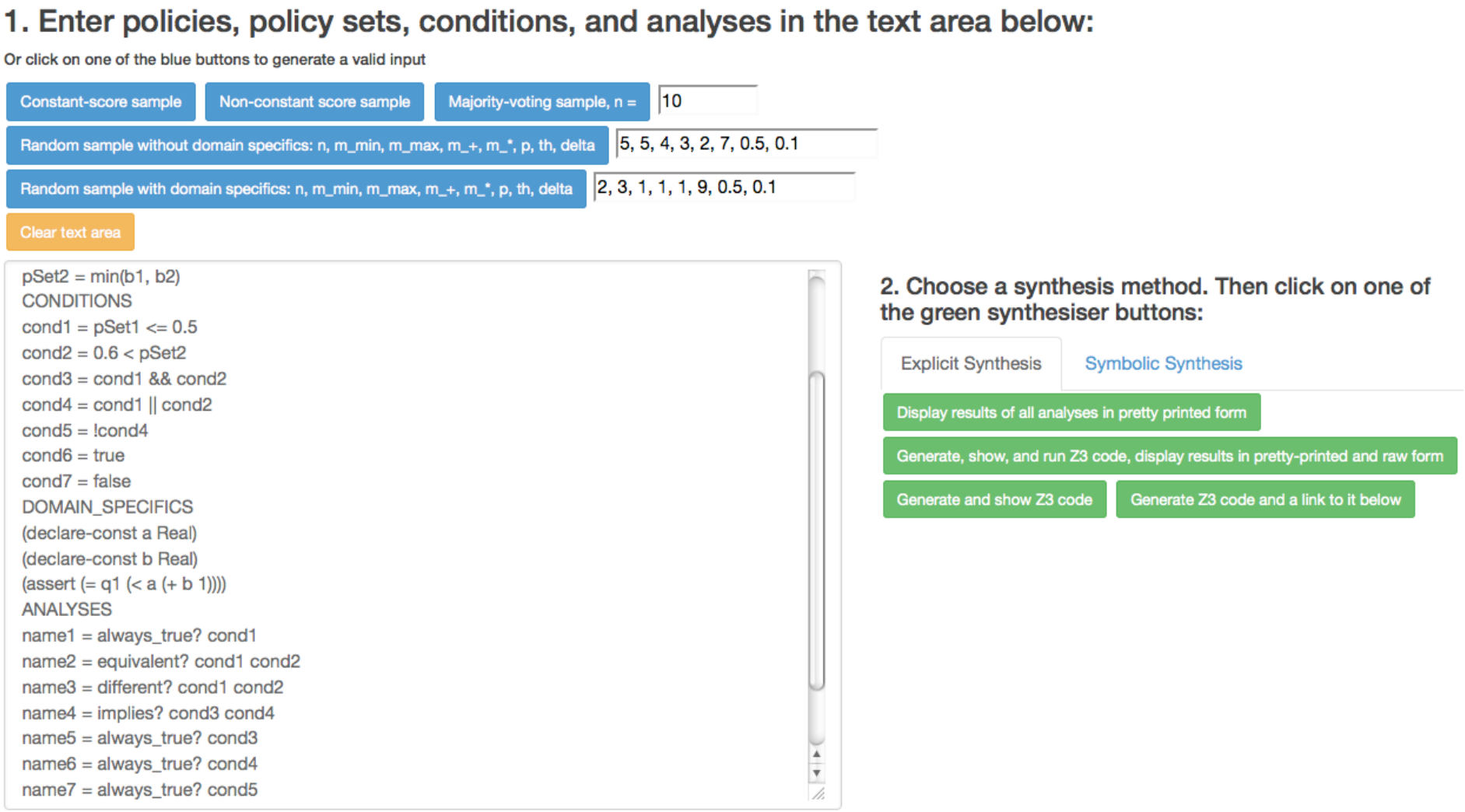
- use simple naming mechanism for all syntactic categories: policies, policy sets, conditions, and analyses

- E.g. we actually bind condition names to formulas in asserts, as in

```
(assert (= cond (and (and (or
q1 q2 q3) (or q1 q2)) (or (and
(not q4) (not q5) (not q6)) (not
false)))))
```


39

DEMO of web app



40

Non-random benchmark

- Majority voting: *pol = +((q1 1/n) (q2 1/n) ... (qn 1/n)) default 0*
- synthesis of *0.5 < pol* generates exponentially many disjuncts in set *M1*
- e.g. for n=27, synthesized formula 0.5 GB **!**
- satisfiability checks easy, size of formula is real issue here: choose(n,ceilint(n/2)) many terms
- explore space-time trade-offs: symbolic synthesis

41

Symbolic synthesis

- Hybrid approach: synthesis of *min* and *max* composition of rules still done explicitly -- linear in input size
- Synthesis of conditions for policy sets makes logical decomposition into conditions for policies explicit
- But synthesis of conditions for policies ***symbolically*** encodes operational semantics with auxiliary real variables

42

Simple example

- a policy composed with *+*
- *b2 = + ((q4 0.1) (q5 0.2) (q6 0.2)) default 0*
- declare real variables *b2_score_qi* for each predicate qi in *b2*
- *assert that these variables have desired operational value*
- exploits that *+* (respectively *) has semantic unit *0* (respectively *1*)

43

Declaring rule effect

b2 = + ((q4 0.1) (q5 0.2) (q6 0.2)) default 0

```
(declare-const b2_score_q4 Real)     ; reals for b2
(assert (implies q4 (= 0.1 b2_score_q4)))
(assert (implies (not (= 0.0 b2_score_q4)) q4))

(declare-const b2_score_q5 Real)
(assert (implies q5 (= 0.2 b2_score_q5)))
(assert (implies (not (= 0.0 b2_score_q5)) q5))

(declare-const b2_score_q6 Real)
(assert (implies q6 (= 0.2 b2_score_q6)))
(assert (implies (not (= 0.0 b2_score_q6)) q6))
```


44

Declaring policy effect

b1 = min (...) ...
b2 = + ((q4 0.1) (q5 0.2) (q6 0.2)) default 0
cond2 = 0.6 < min(b1,b2)

- effect of policy (here b2) is function of condition, here cond2:

```
(declare-const cond2_b2 Bool)
        (assert (= cond2_b2 (or
         (and (< 0.6 0) (not (or q4 q5 q6))) ; default case
         (and (or q4 q5 q6) ; non-default case
                  (< 0.6 (+ b2_score_q4 b2_score_q5
                            b2_score_q6))))))
```


45

Symbolic synthesis

- Z3 input code linear in size of Peal conditions
- space/time tradeoff: e.g. for majority voting
- explicit synthesis could only deal with n=27 for majority voting
- symbolic synthesis generates and verifies equivalent condition for n=5000 in seconds!

46

Outline of presentation

- Motivation
- Peal: pluggable evidence aggregation language
- Generating verification conditions for Peal
- Experimental results
- Future work and conclusions

47

Experiments

- All experiments test: ***Z3 code generation and Z3 analysis both take less than five minutes?***

- Server funded by this Intel project: 24-core, Linux, 48-GB RAM; generated random analyses one single core, here is an example:

```
POLICIES
b0 = max ((q3 0.2602799435758082) (q2 0.8465841883500088) (q1 0.5206939762267451) (q6
0.9230438145042877)) default 0.35448995987981113
b1 = + ((q2 0.5067729289557699) (q1 0.9809171774005019) (q3 0.1957352016220224)) default
0.06811372400781202
b2 = min ((q0 0.3996348599684737) (q6 0.1617558737504945)) default 0.4678754705036007
b3 = * ((q3 0.7832597548785873) (q5 0.5795098955834747)) default 0.4486210823937763
POLICY_SETS
p0_1 = min(b0,b1)
p2_3 = min(b2,b3)
p0_3 = max(p0_1,p2_3)
CONDITIONS
cond1 = 0.50 < p0_3
cond2 = 0.60 < p0_3
ANALYSES
analysis1 = always_true? cond1
analysis2 = always_false? cond2
analysis3 = different? cond1 cond2
```


48

First experiment

- Don't report results on * aggregation of rules in policies for these experiments, see Tech Report for details
- Analyzed *sole* policy with *min* or *max* composition: handle about 2 million rules within five minutes
- *Sole* policy for *+* composition: explicit synthesis can handle around 150 rules within five minutes
- *Sole* policy for *+* composition: symbolic synthesis can handle about six-thousand(!) rules within five minutes

49

First experiment + Domain Specifics

- Repeated first experiment but added domain-specific constraints: linear inequalities, and model of method-call graph with in-degree at most 1
- Analyzed *sole* policy with *min* or *max* composition: handle about 7000 (2-million before) rules within five minutes
- *Sole* policy for *+* composition: explicit synthesis can handle around 130 (150 before) rules within five minutes
- *Sole* policy for *+* composition: symbolic synthesis can handle about two-thousand (6000 before) rules within five minutes

50

Second experiment

- How many policies analyzable in 5 minutes where each policy has x/10 rules and x was 5-minute bound on number of rules from First Experiment (e.g. x = 200,000 for min and max)
- *min* or *max* composition: handle about 50 policies with about 200,000 rules within five minutes
- *+* composition: explicit synthesis can handle around 18,000 policies with 14 rules within five minutes
- *+* composition: symbolic synthesis can handle 24 policies with about six-hundred rules within five minutes

51

Third experiment

- How many policies n with n rules each can we handle within five minutes?
- *min* or *max* composition: handle about 2000 to 3000 policies with that many rules each within five minutes
- + composition: explicit synthesis can handle about 100 policies with 100 rules each within five minutes
- + composition: symbolic synthesis can handle about 160 policies with 160 rules each within five minutes

52

Alice/Bob example

- *cond1* = *0.5* < *pSet* different from *cond2* = *0.6* < *pSet*

- pretty printed explanation extracted from explicit synthesis analysis (colors added manually):

```
Result of analysis [name1 = different? cond1 cond2]
cond1 and cond2 are different

For example, when
highCostTransaction is false
enoughMutualFriendsNormalized is true
lowCostTransaction is true
enoughMutualFriends is true
aFriendOfAliceUnfriendedBob is false
cond2 is false
cond1 is true
numberOfBobsFriends is 5
amountAlicePays is 2.0
numberOfMutualFriends is 5
```


53

Alice/Bob explanation

- *0.5 < pSet* true, *0.6 < pSet* false because
- all predicates in b1 false, so b1 returns 0.6
- first two predicates in b2 true, so b2 returns **at least 0.6**
- don't care about value of *aFriendVouchesForBob*
- so *pSet = min(b1,b2)* returns 0.6

$$b_1 = +((if\ (lowCostTransaction\ 0.3)\ (if\ enoughMutualFriends\ 0.1)\ (if\ enoughMutualFriendsNormalized\ 0.2))\ default\ 0$$

$$b_2 = min((if\ (highCostTransaction\ 0.1)\ (if\ aFriendOfAliceUnfriendedBob\ 0.2)\ (if\ aFriendOfAliceVouchesForBob\ 0.6))\ default\ 1$$

$$cond = 0.5 < min(b_1, b_2)$$

54

Future work

- Comprehensive experimental evaluation
- Case studies that make use of this tool
- Extension to non-constant or negative scores and metrics: symbolic synthesis allows for this
- Independent verification of reported Z3 models, shortening of models for symbolic synthesis
- Implementation of more complex analyses (e.g. computing “threshold spectrum” of policy set)

55

References

M. Huth and J. H.-P. Kuo. Towards verifiable trust management for software execution - (extended abstract). In *Proc. of TRUST*, pages 275–276, 2013.

Crampton, J., Huth, M., Morisset, C.: Policy-based access control from numerical evidence. Tech. Rep. 2013/6, Imperial College London, Department of Computing (October 2013), ISSN 1469-4166 (Print), ISSN 1469-4174 (Online)

M. Huth and J. Kuo. PEALT: A reasoning tool for numerical aggregation of trust evidence. Technical Report 2013/7, Imperial College London, Department of Computing, October 2013. ISSN 1469-4166 (Print), ISSN 1469-4174 (Online).

Thank You

Questions?

57