Neural-Augmented Static Analysis of Android Communication

Jinman Zhao
University of Wisconsin-Madison
jz@cs.wisc.edu

Aws Albarghouthi
University of Wisconsin-Madison
aws@cs.wisc.edu

Vaibhav Rastogi
University of Wisconsin-Madison
vrastogi@cs.wisc.edu

Somesh Jha
University of Wisconsin-Madison
jha@cs.wisc.edu

Damien Octeau
Google
docteau@google.com

Presented by Joshua Learn

Android App Communication Link Discovery

- Applications on the mobile Android platform have the ability to communicate
  - Ex: use external messaging app to send SMS message from within your app
- These communication links can cause huge security vulnerabilities through taking advantage of the user privileges granted to an application
- Problem: detect if communication is possible between two application via static analysis
- Static analysis of large, complex applications is difficult and leads to many reported false positives

Inter-Component Communication (ICC)

- Android Apps communicate with a message system called Inter-Component Communication
- ICC Abuse causes many security vulnerabilities
  - Ex: Bus application broadcasting GPS location to all other applications
  - Ex: SMS spying app disguised as tip calculator
- We want to answer the question: *Can component c communicate with component d?*
- Process is called <u>*link inference*</u>

ICC Overview: Intents and Filters

- Intent used to initiate messages
  - Explicit
    - Target component specified
  - Implicit
    - Functionality specified
      - Action string: action to be performed
      - Set of category strings: additional info about what to do with the intent (ex: “BROWSABLE” - app handling action can open request in a web browser)
      - Set of data fields: data to be acted upon
- Filter used to convey willingness to receive intents
  - Actions: set of strings of accepted intent actions
  - Categories: set of strings of accepted intent categories
  - Data descriptors: descriptions of accepted data fields

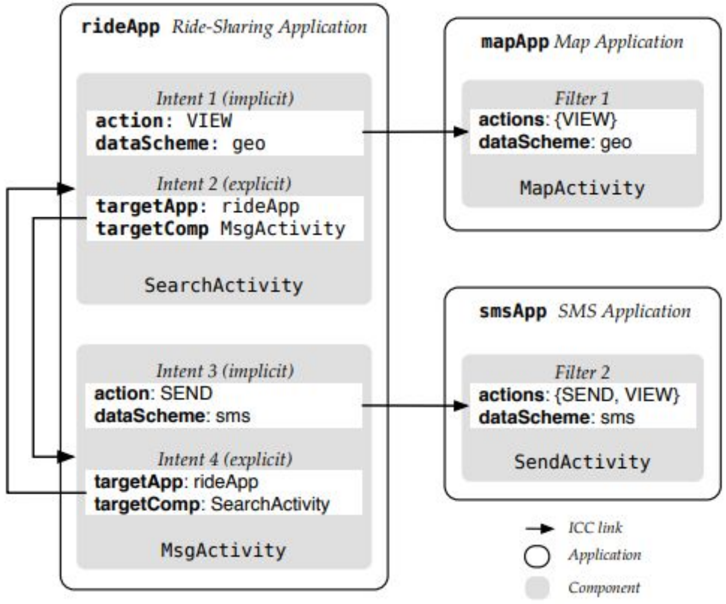
Link Inference

- IC3 is a tool created for Android ICC analysis
- Uses static analysis to infer values of intents and filters
- Inferred values can be used to detect potential links (PRIMO)
- Three possible results:
  - Definite yes: confirmed link between two apps
  - Definite no: confirmed NO link between two apps
  - Maybe: possibility of link exists
- Complex applications yield a high rate of “maybe”s
- Disambiguating “maybe”s is the goal

Relevant Research: PRIMO

- Octeau et al. published probabilistic models for analysing false positives
- Models are handcrafted
- Model creation is months long
- Required deep domain knowledge
- Specific to current Android programming framework
- Includes matching procedure for detecting links between abstract intents/filters

Example




```
public void sendImplicitIntent() {
    Intent intent = new Intent();
    intent.setAction("SEND");
    msg = ... // contains phone # and msg
    intent.setData(msg);
    startActivity(intent);}
```

Code constructing and starting implicit intent

```
<intent-filter>
  <action android:name="SEND"/>
  <action android:name="VIEW"/>
  <data android:scheme="sms"/>
  <category android:name="DEFAULT"/>
</intent-filter>
```

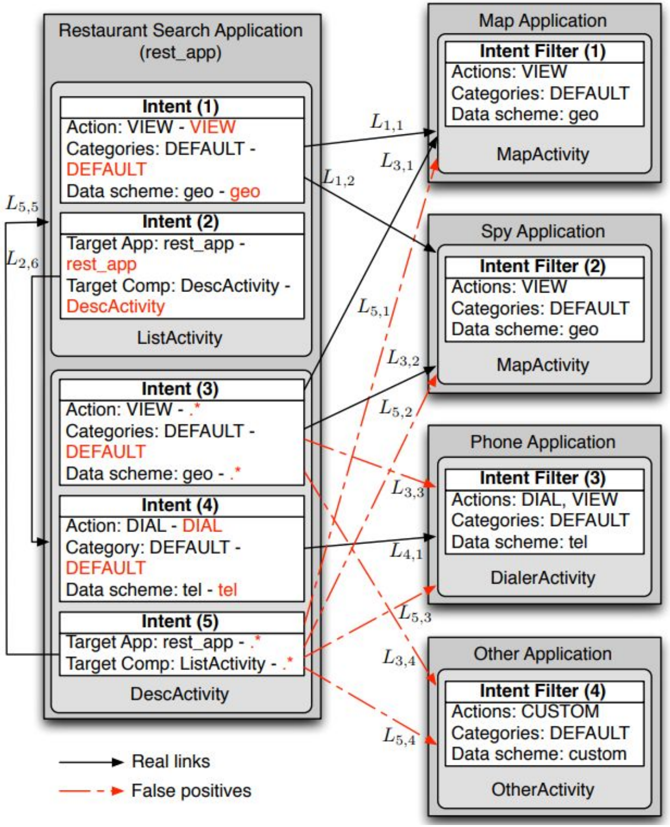
Intent filter for a SMS component

(a) ICC example with three applications

(b) Intent for sending an SMS and associated filter

Figure 2: ICC Example

Vulnerability Example




```java
public void onClick(View v) {
  Location loc =
        LocationManager.getLastKnownLocation("gps");
  Uri query = Uri.parse("geo:" + loc.getLatitude() + ","
        + loc.getLongitude() + "?q=restaurants");
  Intent intent = new Intent("VIEW", query);
  startActivity(intent); }
```

(a) Click handler sending Intent (1) from Figure 3.

```java
public class MapActivity extends Activity {
  public void onCreate(Bundle b) {
    Uri location = getIntent().getData();
    SmsManager.getDefault().sendTextMessage("12345",
          null, location.toString(), null, null); }}
```

(b) Code leaking location data in the spy application from Figure 3.

Formalized Intents and Filters

- Intents
  - Pair *(act, cats)* where
    - $act \in \Sigma^* \cup \{NULL\}$
    - $cats \in 2^{\Sigma^*}$
  - *act* is a string or null representing the action
  - *cats* is the set of strings representing the categories
    - Given no category, *cats* is just the singleton set {“DEFAULT”}
- Filters
  - Pair *(acts, cats)* where
    - $acts \in 2^{\Sigma^*}$
    - $cats \in 2^{\Sigma^*}$
  - *acts* is the set of strings representing the actions
  - *cats* is the set of strings representing the categories

Abstract Intents and Filters

- Static analysis techniques used yield *abstract intents* and *abstract filters*
  - Programmatic creation of intents and filters can lead to many different possibilities at runtime
  - Represent a potentially infinite set of intents/filters through regular expressions
- Abstract versions have same representation structure
  - All strings are regular expressions
  - Ex *act:* (“(.*)SEND”, {“DEFAULT”}) is intent where action has suffix “SEND”
- For every intent/filter in an application, there will be an abstract intent that matches it

Abstract Matching Function

- PRIMO paper offers procedure that infers links:

  $match^{\#}: I^{\#} \times F^{\#} \rightarrow \{0, 1, \top\}$

- Takes an abstract intent and filter
- Yields yes, no, or maybe
- Goal: **disambiguate the maybes**

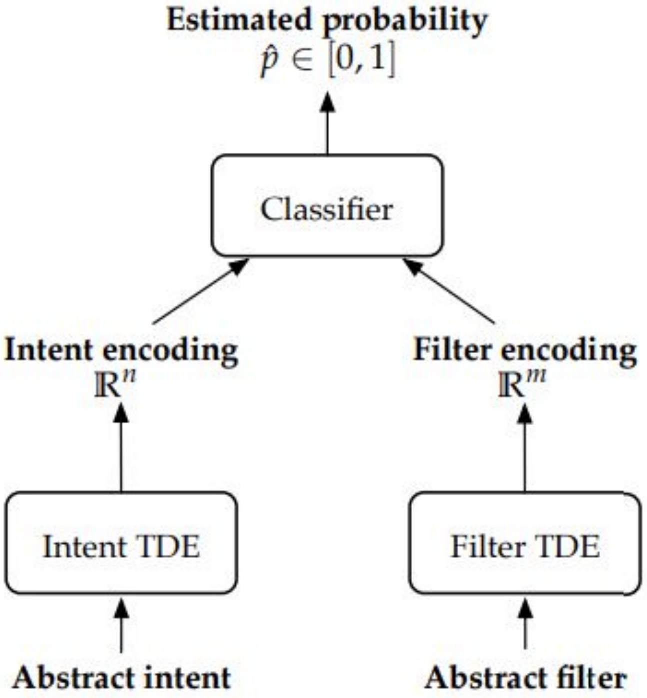
Link Inference as a Classification Problem

- Classifier function:

  $h\colon\ I^{\#} \times F^{\#} \rightarrow [0, 1]$

- Indicates the probability that a link exists ($h(i^{\#}, f^{\#}) = p(y \mid i^{\#}, f^{\#})$)
- Created using Link Inference Neural Network (LINN)
  - Training data: non-maybe labels gathered from static analysis
    - $D = \{\langle(i_1^{\#}, f_1^{\#}), y_1\rangle, \ldots, \langle(i_n^{\#}, f_n^{\#}), y_n\rangle\}$

Link-Inference Neural Network



Type-Directed Encoders

- Need some sort of input representation for abstract intents/filters
- Intents/Filters can be seen as compound data types (sets of strings, unions of strings and null, etc.)
- Type-Directed Encoders recursively encode compound data types
- Encoder of type τ to an n dimensional vector:

  $g: \tau \rightarrow \mathbb{R}^n$

- Encoding functions are Neural Networks jointly trained with the classifier

Encoding Base Types

- Real Numbers
    - already a real number, no encoding needed
- Categories
    - Finite number of possible values (characters, booleans, etc.)
    - Encode *k* categories into n-dim vector by *lookup table* $\boldsymbol{w} \in \mathbb{R}^{nxk}$
    - Encoding for *j*th category is the *j*th column of $\boldsymbol{w}$
    - Achieved using an embedding layer in the neural net
    - Allows us to choose dimensionality of output vector and capture meaning between categories

Encoding Compound Types

- Lists
    - *flat* function
        - trained as CNN or LSTM
- Sets
    - *aggr* function
        - Sum of vectors or Child-sum tree-LSTM
        - No ordering so treated differently than lists
- Products
    - *comb* function
        - MLP or Tree-LSTM unit
- Sums
    - Chooses which encoder to use based on type

Encoding functions

$$\frac{}{\lambda x.\, x \blacktriangleright \mathbb{R}}\ \text{E-REAL} \qquad \frac{enumEnc : \tau_c \rightarrow \mathbb{R}^n}{enumEnc \blacktriangleright \tau_c}\ \text{E-CAT}$$

$$\frac{g : \tau \rightarrow \mathbb{R}^n \quad flat : \mathrm{L}(\mathbb{R}^n) \rightarrow \mathbb{R}^m}{\lambda x.\, flat(map\ g\ x) \blacktriangleright \mathrm{L}(\tau)}\ \text{E-LIST} \qquad \frac{g : \tau \rightarrow \mathbb{R}^n \quad aggr : \mathrm{S}(\mathbb{R}^n) \rightarrow \mathbb{R}^m}{\lambda x.\, aggr(map\ g\ x) \blacktriangleright \mathrm{S}(\tau)}\ \text{E-SET}$$

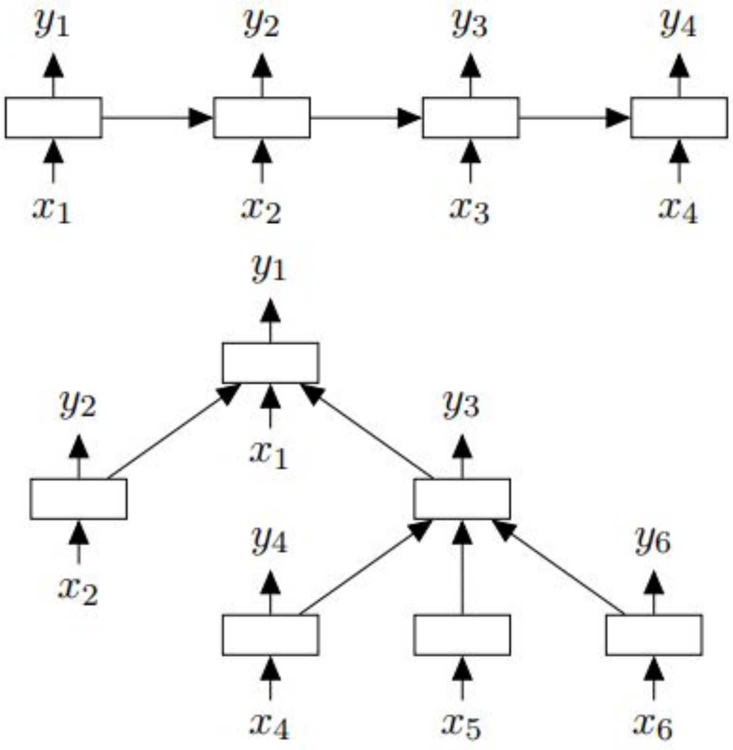
$$\frac{g_1 : \tau_1 \rightarrow \mathbb{R}^n \quad g_2 : \tau_2 \rightarrow \mathbb{R}^m \quad comb : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^l}{\lambda(x, y).\, comb(g_1(x), g_2(y)) \blacktriangleright \tau_1 \times \tau_2}\ \text{E-PROD}$$

$$\frac{g_1 : \tau_1 \rightarrow \mathbb{R}^n \quad g_2 : \tau_2 \rightarrow \mathbb{R}^n}{\lambda x.\, if\ x \in \tau_1\ then\ g_1(x)\ else\ g_2(x) \blacktriangleright \tau_1 + \tau_2}\ \text{E-SUM}$$

Encoding functions

Encoder	Type	Possible differentiable implementations
enumEnc	$\Sigma \to \mathbb{R}^l$	Trainable lookup table (*embedding layer*)
flat	$\mathrm{L}(\mathbb{R}^n) \to \mathbb{R}^m$	CNN / LSTM
aggr	$\mathrm{S}(\mathbb{R}^n) \to \mathbb{R}^m$	*sum* / Child-sum Tree-LSTM unit
comb	$\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^l$	Single-layer MLP / binary Tree-LSTM unit

Tree-LSTM



Intents

$$(L(\sum) + \boldsymbol{\Omega}) \times S(L(\sum))$$

Intents

$$(\underline{L(\sum)} + \boldsymbol{\Omega}) \times S(L(\sum))$$

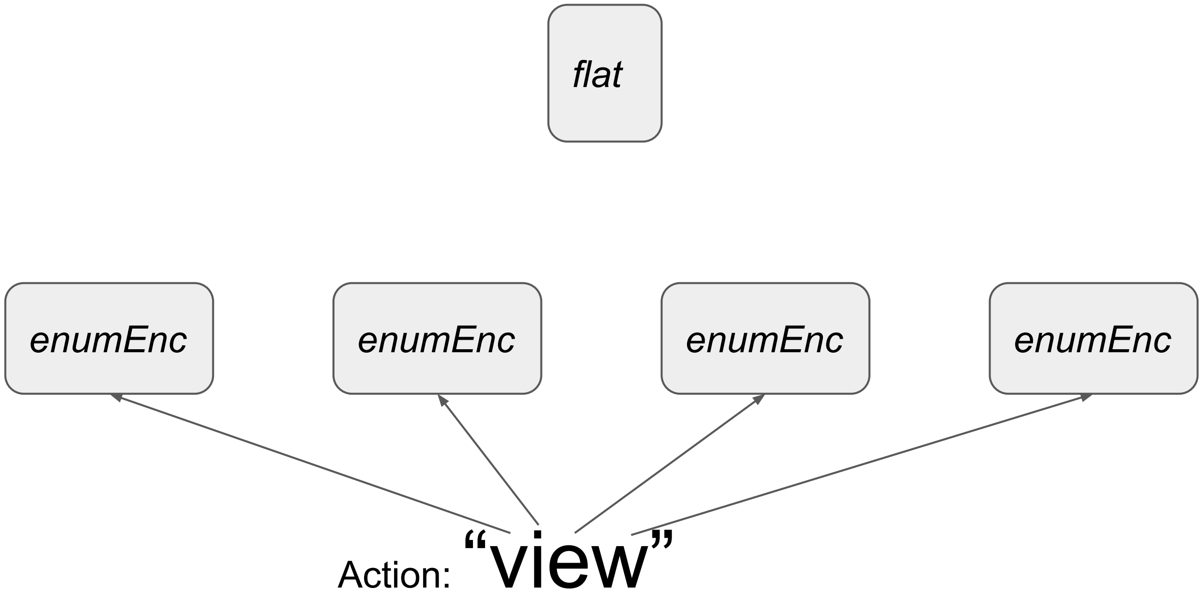
flat

enumEnc *enumEnc* *enumEnc* *enumEnc*

Action: “view”

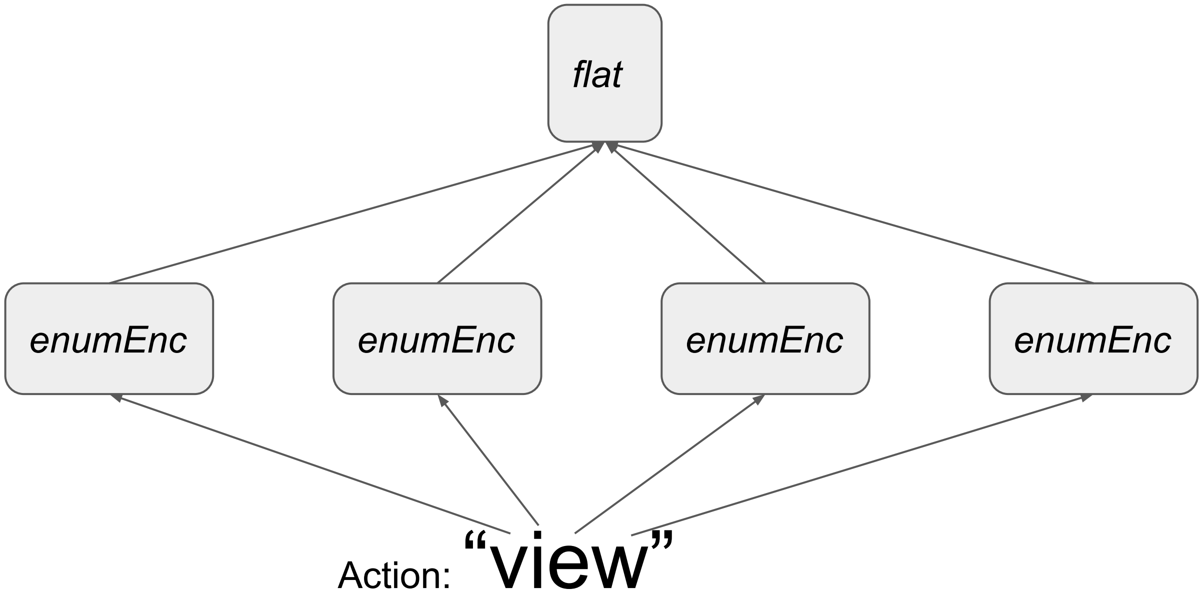
Intents

$$(\underline{\mathbf{L}(\Sigma)} + \Omega) \times S(L(\Sigma))$$



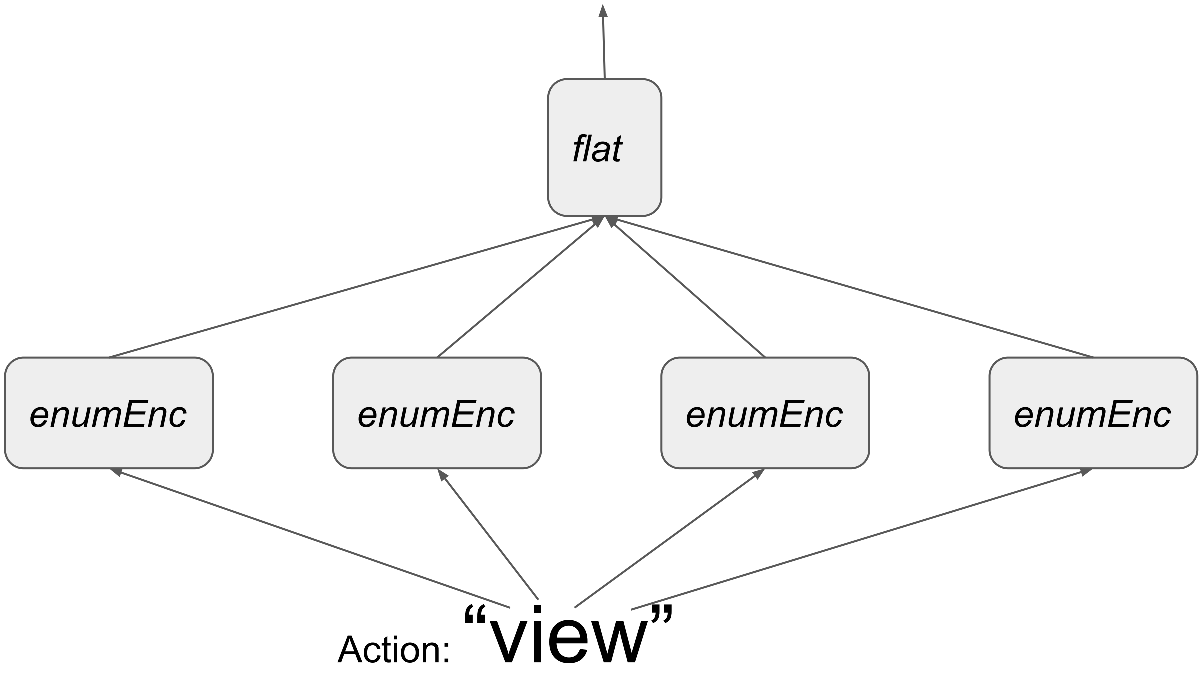
Intents

$$(\underline{\mathbf{L}(\Sigma)} + \boldsymbol{\Omega}) \times S(L(\Sigma))$$



Intents

$$(\underline{L(\Sigma)} + \Omega) \times S(L(\Sigma))$$



Intents

$$(L(\Sigma) + \Omega) \times \underline{\mathbf{S(L(\Sigma))}}$$

aggr

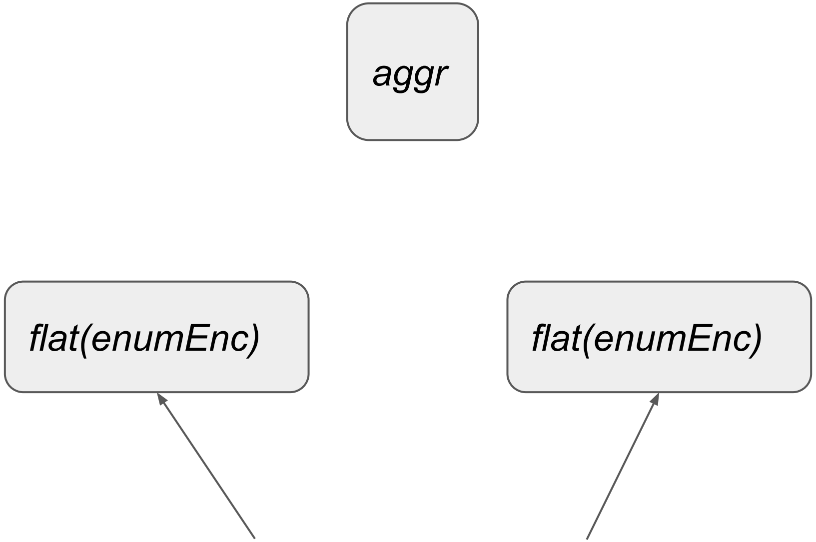
flat(enumEnc)

flat(enumEnc)

Categories: {“BROWSABLE”, “OTHER”}

Intents

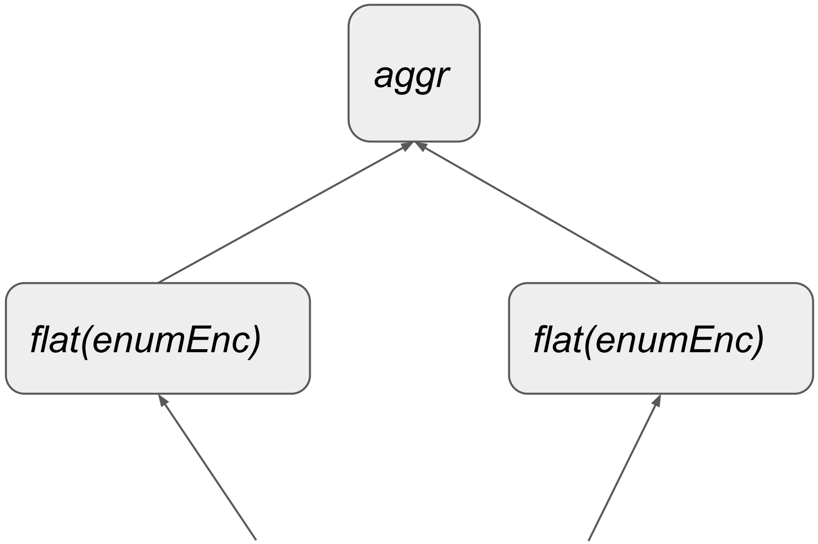
$$(L(\sum) + \Omega) \times \underline{\mathbf{S(L(\sum))}}$$



Categories: {“BROWSABLE”, “OTHER”}

Intents

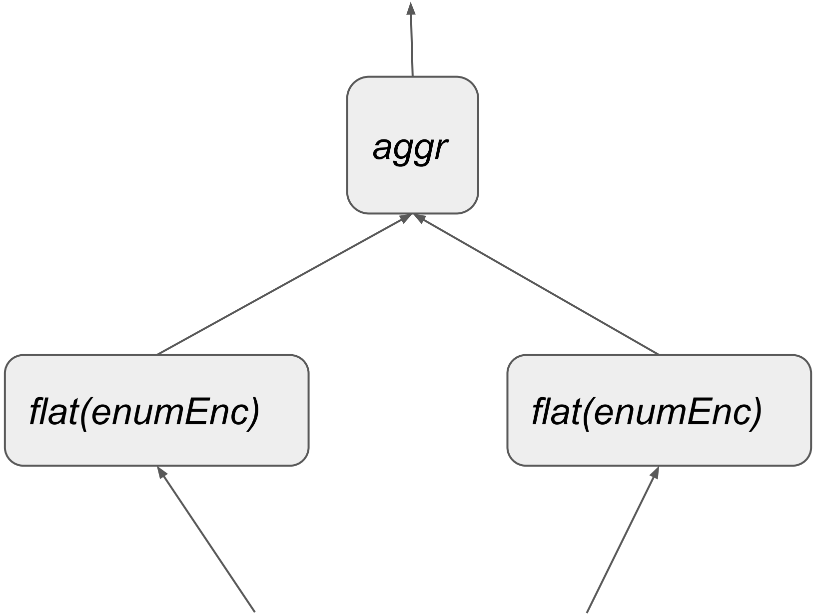
$$(L(\sum) + \mathit{\Omega}) \times \underline{\mathbf{S(L(\sum)}})$$



Categories: {“BROWSABLE”, “OTHER”}

Intents

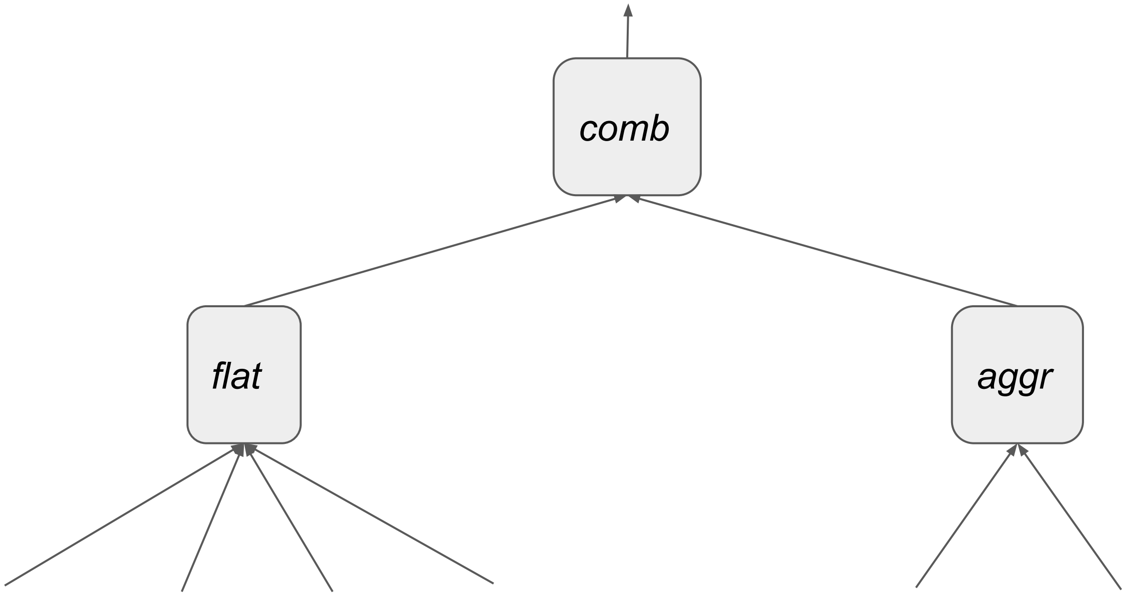
$$(L(\sum) + \Omega) \times \underline{\mathbf{S(L(\sum)}})$$



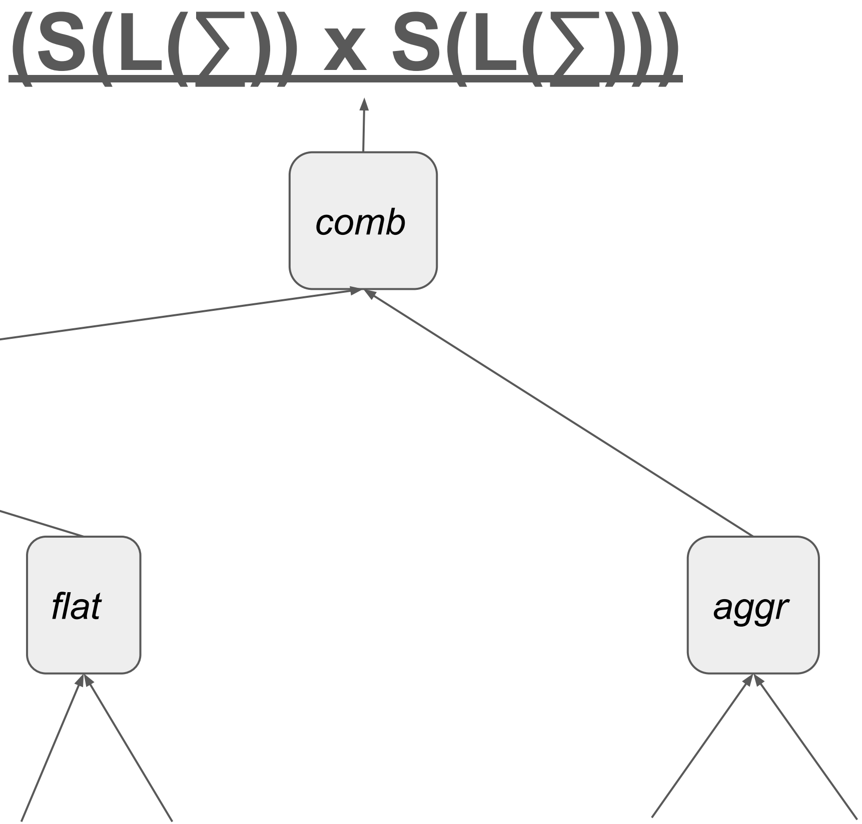
Categories: {“BROWSABLE”, “OTHER”}

Intents

$$\underline{(L(\sum) + \Omega) \times S(L(\sum))}$$



Filters



Different Implementations

Table 2: Instantiations of TDE parameters

		TDE parameters			
Instantiation	Type	*enumEnc*	*flat*	*aggr*	*comb*
str-RNN	L(Σ)	*lookup*	RNN	-	-
str-CNN	L(Σ)	*lookup*	CNN	-	-
typed-simple	full	*lookup*	CNN	*sum*	1-layer perceptron
typed-tree	full	*lookup*	CNN	Tree-LSTM	Tree-LSTM

Hyperparameters

Hyperparameter		Choice
Lookup table	dimension	16
CNN	kernel sizes	$\langle 1, 3, 5, 7 \rangle$
	kernel counts	$\langle 8, 16, 32, 64 \rangle$
	activation	relu
	pooling	max
RNN (LSTM)	hidden size	128
1-layer perceptron	dimensions	64
	activation	relu
Multilayer perceptron	dimensions	$\langle 16, 1 \rangle$
	activation	$\langle \text{relu}, \sigma \rangle$

Implementation Details

- Python with Keras (TensorFlow backend)
- Cross Entropy loss function (model outputs a probability)
- RMSprop variation of stochastic gradient descent
- Relu used for all activation functions
- LINN trained on GPU

Experimental Setup

- PRIMO corpus used for dataset
  - 10,500 Android Apps from Google Play
- IC3 + PRIMO abstract matching for static analysis
  - Provides dataset with must/may link labels
- Synthetic *may* links used for training and testing the model
- Model trained on a sampled subset of links
  - Using all available data too costly
  - Number of links inferred quadratic to the number of intents/filters
  - Sampling balanced between positive and negative labels
- Testing done only on *may* links

Simulating Imprecision

- Ground truth of *may* labels is unknown
- Synthetic *may* labels created by introducing imprecision to *must* links
  - Ex: add “(.*)” to the beginning of a string
  - Technique used by Octeau et al. when creating PRIMO
- First study empirical distribution of imprecision from corpus
  - Add imprecisions guided by the distribution of imprecision observed

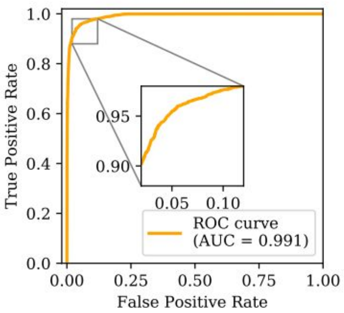
Evaluation Metrics Used

- F1 Score
  - Measure of predictor’s false-negative and false-positive rates
  - Perfect precision/recall has F1 score of 1
- ROC Curve
  - Plot of true positive against true negative rate
  - Perfect model has area under curve of 1
- Kruskal’s γ
  - Correlation between ranking computed by model and ground truth
  - Useful because we want to use model to present results in order of likelihood for programmers to observe

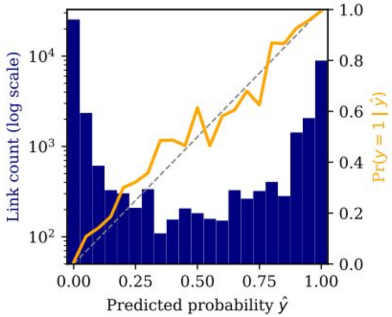
Results

Table 4: Summary of model evaluations

Instantiation	# Parameters	Inference time (μs/link)	Testing γ	Testing F1	AUC	Entropy of $\hat{y}$	$Pr(y = 1 \mid \hat{y} > 0.95)$	$Pr(\hat{y} > 0.95)$
str-RNN	154,657	2220	0.970	0.891	0.975	3.002	0.980	0.089
str-CNN	**27,409**	**57**	0.988	0.917	0.988	2.534	**0.998**	0.139
typed-simple	142,417	157	0.989	0.920	0.988	2.399	0.996	0.173
typed-tree	634,881	171	**0.992**	**0.931**	**0.991**	**2.220**	0.994	**0.200**



(a) Receiver operating characteristic (ROC)



(b) Distribution of predicted link probabilities

Figure 5: Detailed results for the typed-tree instantiation

Observations

- Typed-tree yields the best overall results
- Typed-simple is still slightly better than Str-CNN
- str-CNN has the fastest inference time and best probability of true-positive among highly ranked links
- str-CNN may be preferable but market scale analysis would benefit from slight increases in accuracy
- 10 epochs of training take <20 minutes for all except str-RNN
  - Average computer used
    - Intel i7-6700 (3.4 GHz)
    - 32GB RAM
    - 1TB SSD
    - Nvidia GeForce GTX 970 GPU
- Most complex model has only 5.6MB storage cost

Str-CNN Characteristics

{"action": "NULL-CONSTANT", "categories": null}
{"actions": ["NULL-CONSTANTPOP_DIALOG", "NULL-CONSTANTPUSH_DIALOG_(.*)", "(.*)REPLACE_DIALOG_(.*)", "APP-00489869YB964702HUPDATE_VIEW"], "categories": null}

{"action": "NULL-CONSTANTREPLACE_DIALOG_(.*)", "categories": null}
{"actions": ["(.*).CLOSE"], "categories": null}

{"action": "(.*)", "categories": null}
{"actions": ["android.media.RINGER_MODE_CHANGED", "sakurasoft.action.ALWAYS_LOCK", "android.intent.action.BOOT_COMPLETED"], "categories": null}

{"action": "(.*)LOGIN_SUCCESS", "categories": null}
{"actions": ["NULL-CONSTANTLOGIN_FAIL", "NULL-CONSTANTCREATE_PAYMENT_SUCCESS", "(.*)FATAL_ERROR", "(.*)CREATE_PAYMENT_FAIL", "NULL-CONSTANTLOGIN_SUCCESS"], "categories": null}

{"action": "APP-00489869YB964702HREPLACE_DIALOG_(.*)", "categories": null}
{"actions": ["APP-00489869YB964702HLOGIN_FAIL", "APP-00489869YB964702HCREATE_PAYMENT_FAIL", "NULL-CONSTANTCREATE_PAYMENT_SUCCESS", "(.*)FATAL_ERROR", "NULL-CONSTANTLOGIN_SUCCESS"], "categories": null}

{"action": "com.joboevan.push.message.(.*)", "categories": null}
{"actions": ["com.joboevan.push.message.NULL-CONSTANT"], "categories": null}

{"action": "", "categories": ["(.*)"]}
{"actions": ["com.dreamware.Hells_Kitchen.CONCORRENTE"], "categories": ["android.intent.category.DEFAULT"]}

{"action": "(.*)", "categories": null}
{"actions": ["android.intent.action.MEDIA_BUTTON", "com.ez.addon.MUSIC_COMMAND", "android.media.AUDIO_BECOMING_NOISY"], "categories": null}

Figure 6: Explaining individual instances

Str-CNN Characteristics

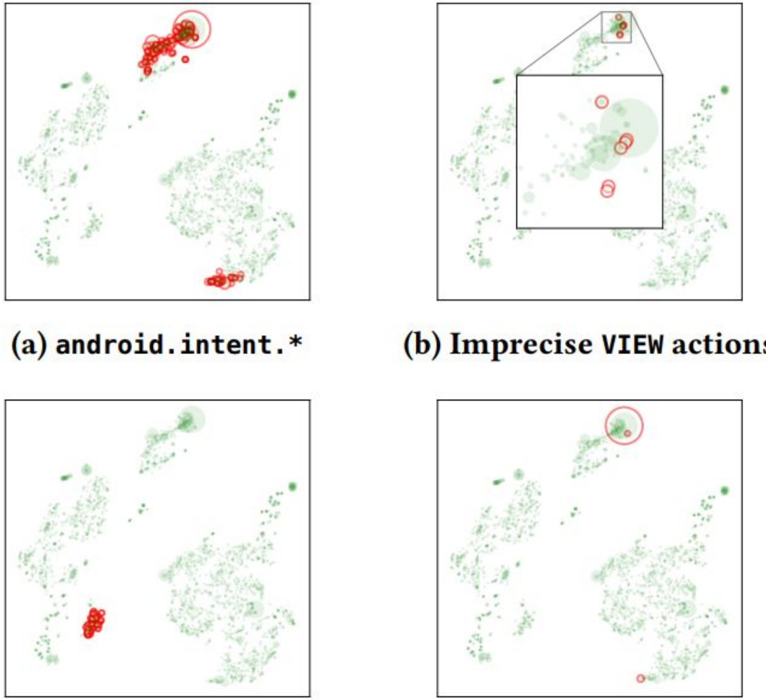
- Tested input strings to see what patterns kernels are picking up
- Important segments seem to be picked up
  - conv1d_size5:14 kernel activated on “.*”
  - conv1d_size5:3 kernel activated on “null”
  - conv1d_size7:0 kernel activated on “VIEW”

Table 5: Some CNN kernels and their top stimuli

conv1d_size5:14		conv1d_size5:3		conv1d_size7:0	
segment	activation	segment	activation	segment	activation
(.*)R	1.951	null}	3.796	TAVIEWA	3.704
(.*)u	1.894	null,	2.822	n.VIEW"	3.543
(.*)t	1.893	sulle	2.488	y.VIEW"	3.384

Typed-Simple Visualization

- t-SNE non-linear dimensionality reduction
  - Similar objects mapped to nearby points
  - Dissimilar objects mapped to distant points
- Six imprecise versions of VIEW captured
  - (.*) occurs at different points in the string
  - Imprecision reflected spatially
- DEFAULT, (.*), null categories all in close proximity



(a) `android.intent.*` (b) Imprecise `VIEW` actions (c) `dev*.app*.*.FEED*` (d) `DEFAULT`, total imprecise and null categories

Figure 7: Intent encodings visualized using t-SNE

Possible Concerns/Invalidities

- Tested on synthetic *may* links
  - Follows empirical distribution of imprecisions
  - Might not capture all meaning in real world data
- Neural network setup is complex
  - Difficult to know if relevant features are being captured or the NN is getting “lucky”
  - Best performing model has many parameters and may be overfitting
- Performance is not significantly better than plain str-CNN
  - More time invested may discover a simpler and better way to embed intents/filters

Future Work

- Main novelty of this paper was Type-Directed Encoders
  - Framework for composing neural networks
  - Applies nicely to the problem of link inference in the Android domain
- TDE could be applied to other contexts that exhibit a structure of data composed of subtypes

References

- https://arxiv.org/pdf/1809.04059.pdf
- http://delivery.acm.org/10.1145/2840000/2837661/p469-octeau.pdf?ip=160.39.169.169&id=2837661&acc=CHORUS&key=7777116298C9657D%2ECCAFA7F43E96773E%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1554088993_7d6fdb889b8c94e87c503e4666f2cb7a
- https://arxiv.org/pdf/1503.00075.pdf
- https://developer.android.com/guide/components/intents-filters