Reverse Architecting Software Binaries

HCSS 2024 – May 8, 2024

Greg Nelson (gnelson@grammatech.com)
Coauthor Denis Gopan (gopan@grammatech.com)

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Reverse Architecting

- Goals
  - Recover *high level architecture* from binary
  - Compare *design* with *implementation*
- Outcomes
  - Advanced state of the art in recovery algorithms
  - Developed novel algorithm for des-impl comparison
  - Tools effectively applied in DARPA ARCOS program

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Outline

- Value of reverse architecting
- Role in reverse-engineering toolchain
- Componentization research
- Design-Implementation mapping research
- Conclusions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Defining the Problem

- Assurance-adjacent example
  - Commercial device (e.g., PLC) in critical industry
  - Want to evaluate fitness (safety & security)
  - Vendor doesn't offer source code, keeps implementation details proprietary
  - How can you assess based on *binary only?*

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Goal: Recover *High-Level Architecture*

- Disassembly / decompilation (Ghidra): too low level
  - Recovering functions is not enough
- What we actually want:
  - *Components:* subsystems, libraries, modules, classes
  - *Relations:* containment, communication, etc.
    - Module A is comprised of modules B, C, and D
    - Module A calls module B's routines
    - Modules A and B share common data
    - …

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Goal: Align *Design* to *Implementation*

- When you have design or specifications (or, if you are willing to define them)
  - How do they match the implementation?
  - Traceability can show you
    - Where is security-critical code?
    - Which subsystem in the design is impacted by a CWE?
    - Was code included that was not in the specification?
    - Are there requirements which are not implemented?

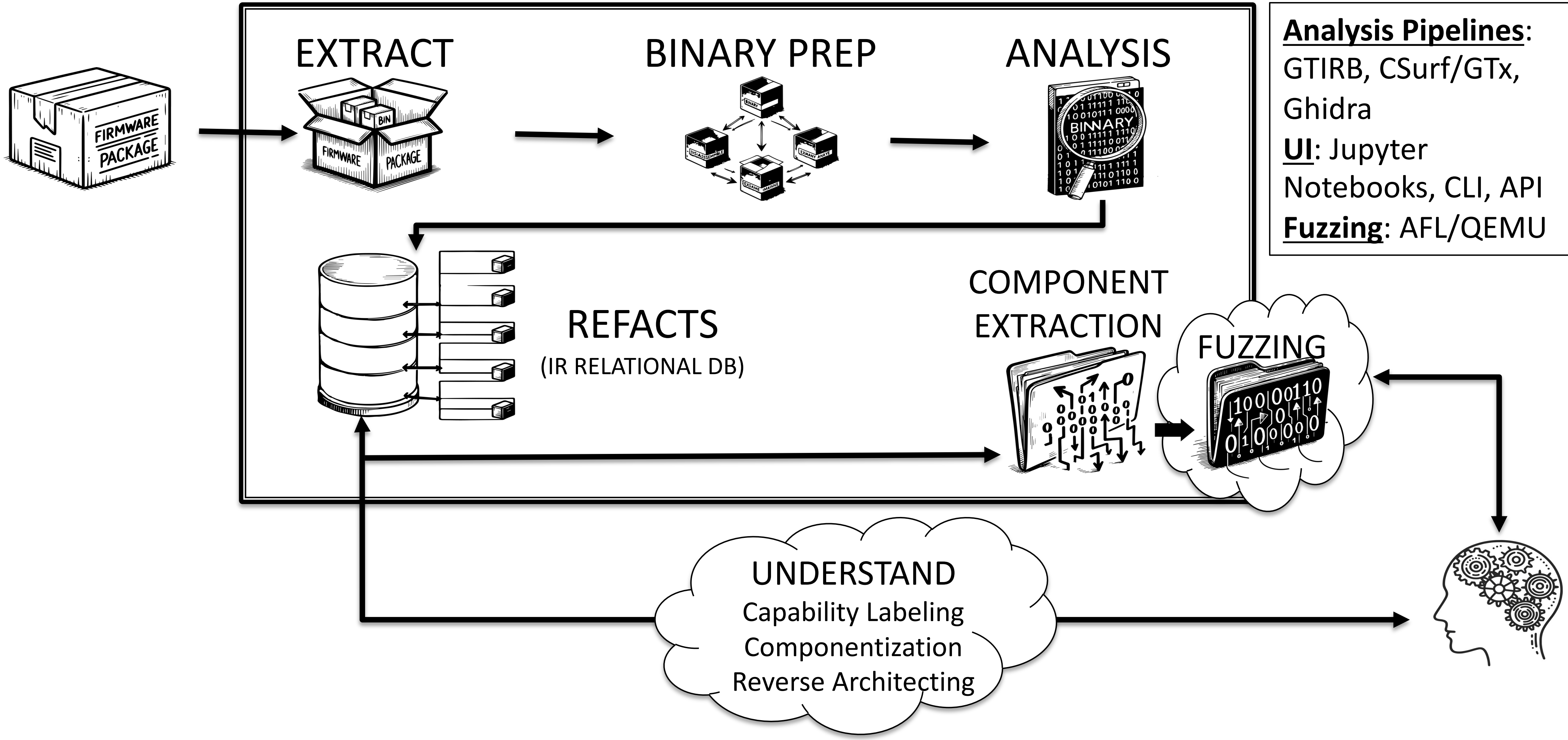
© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Outline

- Value of reverse architecting
- **Role in reverse engineering toolchain**
- Componentization research
- Design-Implementation mapping research
- Conclusions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

REAFFIRM Toolchain Overview



Analysis Pipelines: GTIRB, CSurf/GTx, Ghidra
UI: Jupyter Notebooks, CLI, API
Fuzzing: AFL/QEMU

GRAMMATECH

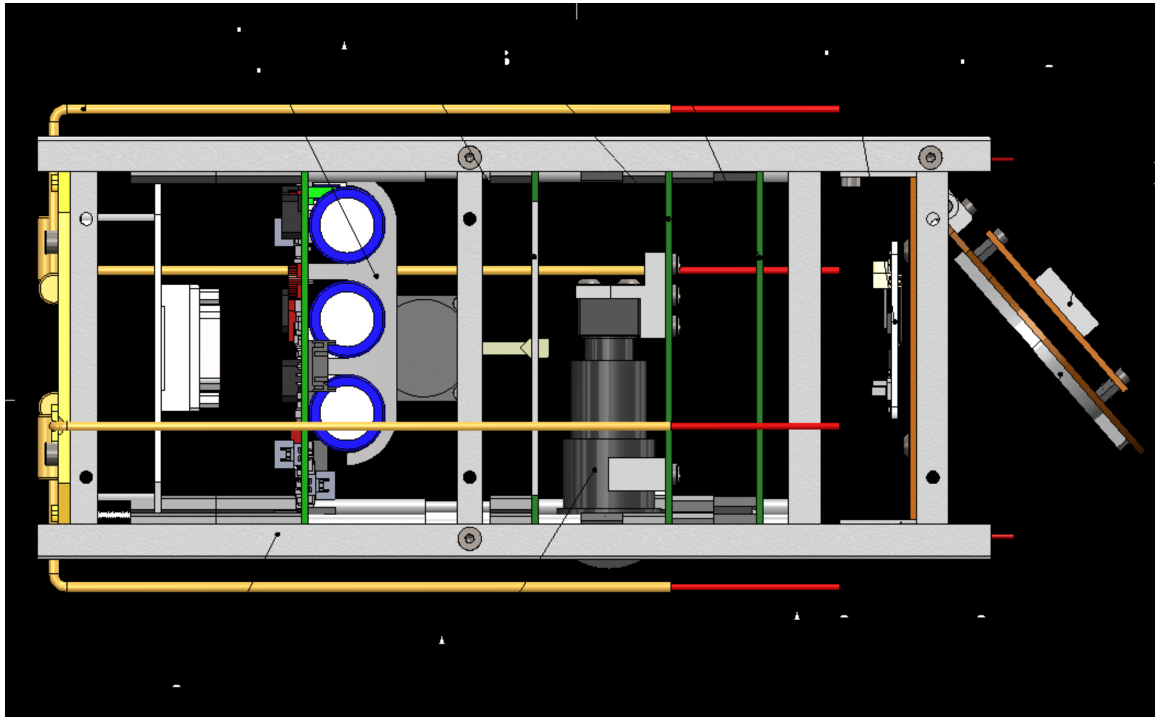
© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

UPSat (Example) Introduction

UPSat

Most examples drawn from "UPSat"

- University of Patras Satellite
- Small "CubeSat" from QB50 project
- Four embedded STM32 processor boards
- Open source, C-language
- Mostly bare-metal
- No system design artifacts



© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Outline

- Value of reverse architecting
- Role in reverse engineering toolchain
- Componentization research
- Design-Implementation mapping research
- Conclusions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

What is a “Component”?

- A: Component Grouped by *Layer*
  - Single level of abstraction
  - Possibly low similarity of purpose
  - *Example:* hardware abstraction layer (HAL), Ethernet driver
- B: Component Grouped by *Function*
  - Single conceptual “purpose”
  - May represent several “layers” of abstraction
  - *Example:* subsystem, library (network stack, encryption, …)
- Design could be A, B, or a mixture of both
- Sometimes reflected in language paradigms (more later…)

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Binary Componentization

- Identify related functionality
- Create containment tree
  - Binary, at root, contains everything
  - Functions at leaf nodes (treated as “atoms”)
  - Components (internal nodes) are inferred
- We also infer communication (not discussed)

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Evaluating Componentization

- Related question: What is a good design?
  - Different designers take different approaches
  - Only agreement in literature: this is *hard* [1, 3]
- To evaluate performance, need a “ground truth”
  - Must be easy to generate for test binaries
  - Must have some similarity to developer’s design
  - Use *map file* as simple approximation (as in [4])

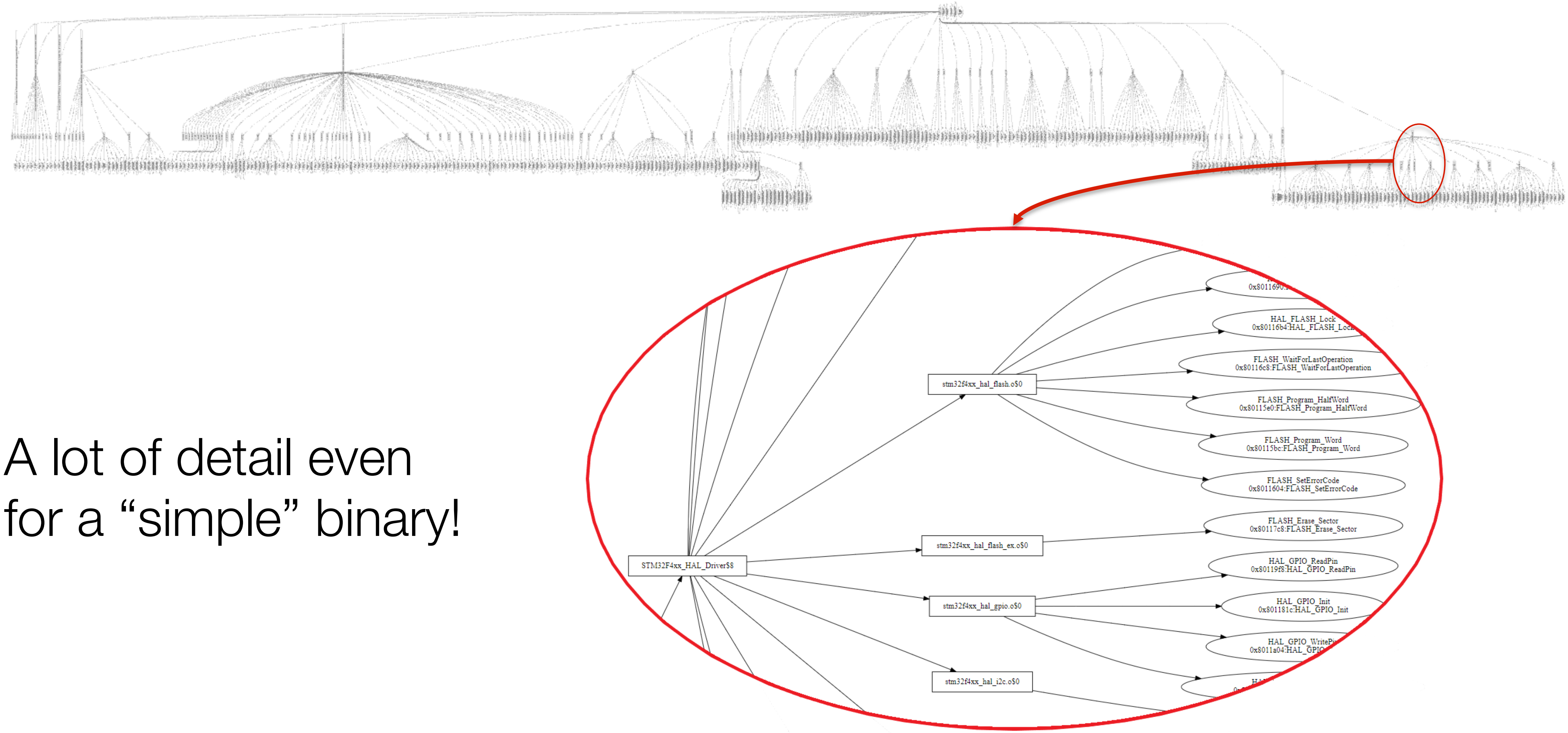
© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Evaluating Componentization

- Map file supplies these testable properties:
  - Grouping of functions in source modules
  - Organization of source modules (from path names)
  - Libraries – functions and object files
- Measurement ambiguity: libraries
  - Often one function per compilation unit (source file); allows linker to discard unused library functions
  - Do we treat *CU* or *Library* as the “correct” component?
  - Decision: Evaluate algorithms both ways

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Evaluating Componentization



A lot of detail even for a “simple” binary!

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Componentization Algorithms

- Allow for multiple algorithms
- Each should produce comparable graph structure
- Currently implemented:
  - Map file: use for ground truth; also useful as end-user tool
  - Compilation Units: crude (but stable) linear partitioning
  - GT-BCD: graph clustering based on multiple features

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Quick Comparison of Algorithms

Class	Feature	CompUnit	BCD (replic.)	GT-BCD
Code Locality	Boolean Adjacency	X	0.237	0.126
	Weighted proximity			0.100
Function calls	Direct	X	0.362	0.152
	Sibling			0.218
Data references	Sibling		0.400	0.153
Naming (if avail.)	Name Prefix	X		0.250
UPSat Result (Precis%|Recall%)	Matched by CU	40.4|91.7	25.7|49.2	48.8|54.6
	Matched by Library	75.0|35.4	42.7|17.0	65.8|15.3

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Componentization: CompUnits Algorithm

- Assume compilation units are contiguous, look for boundaries
- Relies heavily on proximity and naming
  - Proximity: CU is contiguous (but may have some loosely related functions); sliding window
  - Naming: tools (C++) group classes and name spaces; developers often use common prefixes ("HAL_GPIO_*")
- Works well for some binaries (incl. UPSat)
- Poor performance w/o names, or if binary reordered

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Componentization: GT-BCD Algorithm

- Inspired by BCD work of Karande et al. [4]
- Superimposed, weighted subgraphs
  - Allows for multiple, individually weighted features
  - Easy to add/experiment with new features
- Community detection algorithm
  - Agglomerative clustering [2,6,7]
- Original targets: C++ binaries for Windows, Linux
- Original BCD less effective for *embedded binaries*

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Programming Language Bleed-through

- Karande's BCD works well for OOP (C++, Ada)
- *Does not* work well for pure procedural (C, assy)
- CG and DRG efficacy improved by OOP's VFT
- Strict adjacency too restrictive for some code
- **GT-BCD** explored other compensating features

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

New GT-BCD Features

- Window adjacency, proportional adjacency
  - Helps with compile/link optimization, small ‘helpers’
- Naming (prefix, edit distance) *when available*
  - Weak by itself but plays well with others
- Sibling Calls
  - Karande BCD has “A→C ⇒ A~C”
  - “Signal” for calls *within* module, but “noise” for calls *into* module, e.g., API
  - GT-BCD adds “A→C & B→C ⇒ A~B”
- Weighting: Used gradient search to optimize

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

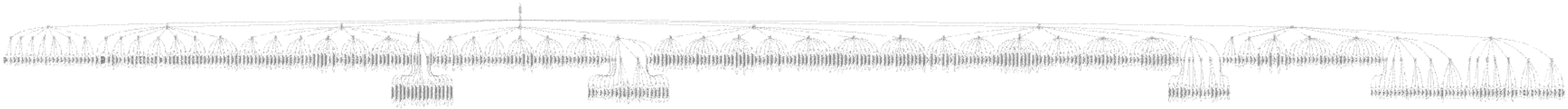
Graph Clustering Approaches

- ~~Newman algorithm~~: original, **slow** $O(e^*v+v^2)$
- Leiden algorithm: shown to avoid certain non-optimal partitions, and empirically fast
- Clauset-Newman-Moore (CNM): fast on sparse networks ($e \approx v$ and $d \approx \log v$), dendrogram result
- Variants to specify number of clusters or layers
- Allows tuning if architecture is known
- All of these are non-deterministic (greedy algs.)

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

GT-BCD Recreates Design from Binary

- Design structure can be recovered
- Not identical to map, but plausible similarity
- Quantitative similarity measurement non-trivial
  - Grouping, levels, over/under-splitting, etc.



© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Outline

- Value of reverse architecting
- Role in reverse engineering toolchain
- Componentization research
- Design-Implementation mapping research
- Conclusions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Why Mapping Matters

- If you already have the design, why map?
  - Requirements traceability beyond source into binary
  - Vulnerability traceability back to design
  - Detection of code *not* traceable to requirements
  - Software Reflexion Models [5] shows many uses… but it assumed that the *map* had to be done by hand

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Design-to-Implementation Mapping

- This stage assumes design or requirements exist
- How do design entities $\mathcal{D}$…
  map to implementation entities $\mathcal{I}$?
  - Assume we know tree roots: “system” == “binary”
  - Assume $\mathcal{D}$ leaves are similar to $\mathcal{I}$ leaves
  - Natural language content ($\mathcal{D}$ text, $\mathcal{I}$ strings/capabilities)
  - Structural similarity (hierarchy, branching)
  - Structural content (e.g., “API consists of 8 functions”)

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Mapping Challenges

- From specification
  - Stale or reconstructed specification: likely inaccurate
  - Under-specification: e.g., no explicit mention of libraries
- From componentization
  - *Over-splitting:* binary modules are too small
  - *Under-splitting:* binary modules are too large
  - *Instability:* non-deterministic clustering algorithms
  - *Binary obfuscation:* shuffled or self-modifying code
- From either input
  - Components with zero semantic information (all equivalent)

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Mapping Approaches

- "Brute force" graph isomorphism
  - Too rigid for real-world mappings; worst-case intractible
- Simple tree traversal
  - Easy top-down process, but fails to incorporate bottom-up info
- Models of human analogical reasoning
  - Examples: SME, ACME, Sapper
  - Core principle: combining semantic and structural inputs
  - AI-driven approaches that retain explainability

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

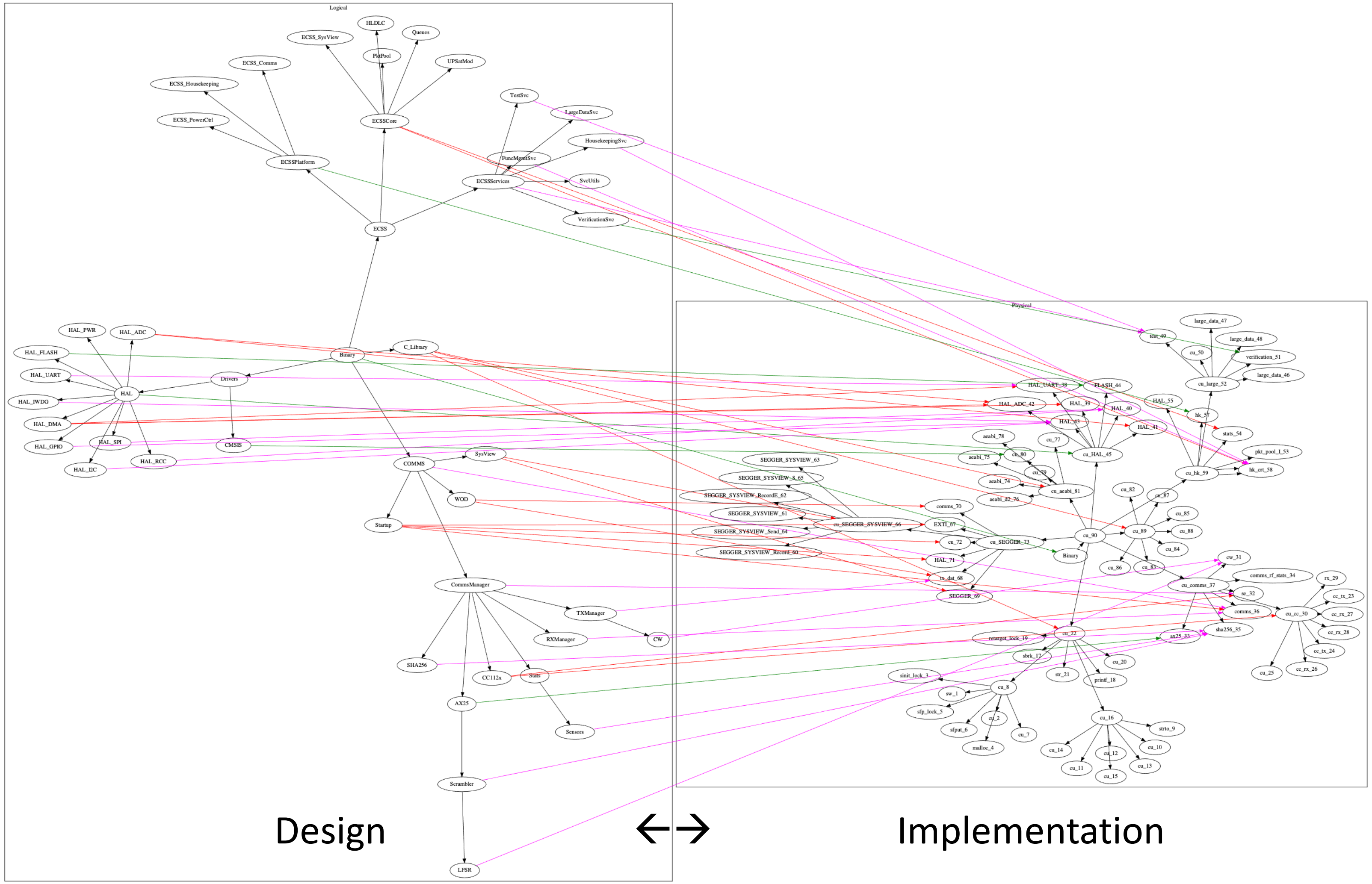
ACRE Algorithm

- Algorithm for Component Reflexion Estimation
  1. Graph preparation
  2. Semantic matching of $\mathcal{D}$ to $\mathcal{I}$ (structure-constrained)
  3. Tracing parentage of each $\mathcal{D}$ and $\mathcal{I}$ component
  4. Propagate structural/analogical relations
  5. Combine semantic and structural confidence
  6. Best-first, implementation-driven map assignment

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Example UPSat Mapping



Design ←→ Implementation

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

$\mathcal{D}$-to-$\mathcal{I}$ Mapping Takeaways

- The process is non-trivial
- Quality of componentization matters
- Structural and semantic information both needed
- Cognitive models of analogy provide insights
- Can provide useful ability to:
  - Determine design-level impact of binary CWEs
  - Identify compiled-in code not present in

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Outline

- Value of reverse architecting
- Role in reverse engineering toolchain
- Componentization research
- Design-Implementation mapping research
- Conclusions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Conclusions/Future Work

- Recovering architecture from binary is
  - Valuable
  - Complex
  - Achievable
- Possible future research
  - Other architectural relations: communication, specialization, etc.
  - Deeper semantic recovery
  - Better methods for evaluating ground truth
  - Use of LLMs in structure matching

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Questions

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

References

1. Anquetil, N. and Lethbridge, T.C., 1999. “Experiments with clustering as a software remodularization method.” In *Proceedings of Sixth Working Conference on Reverse Engineering* (Cat. No. PR00303) (pp. 235-255). IEEE. https://doi.org/10.1109/WCRE.1999.806964
2. Clauset, A., Newman, M.E.J., and Moore, C., 2004. “Finding community structure in very large networks.” In *Physical Review E,* 70(6), p. 066111. https://doi.org/10.1103/PhysRevE.70.066111
3. Clayton R, Rugaber S, Wills L., 1998. “On the knowledge required to understand a program.” In *Proceedings of Fifth Working Conference on Reverse Engineering* (Cat. No. 98TB100261) (pp. 69-78). IEEE. https://doi.org/10.1109/WCRE.1998.723177
4. Karande, V. et al., 2018. “BCD: Decomposing binary code into components using graph-based clustering.” In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (pp. 393-398). https://doi.org/10.1145/3196494.3196504
5. Murphy, G.C., Notkin, D. and Sullivan, K., 1995. “Software reflexion models: Bridging the gap between source and high-level models.” In *Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering* (pp. 18-28). https://dl.acm.org/doi/pdf/10.1145/222124.222136
6. Newman, M.E., 2004. “Fast algorithm for detecting community structure in networks.” *Physical review E*, 69(6), p. 066133. https://doi.org/10.48550/arXiv.cond-mat/0309508
7. Traag, V.A., Waltman, L. and Van Eck, N.J., 2019. “From Louvain to Leiden: guaranteeing well-connected communities.” *Sci. Rep.* 9, 5233. https://doi.org/10.1038/s41598-019-41695-z

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Backup Slides

- REAFFIRM summary
- Tables of Feature Performance
- Attempted features that were ruled out
- Mapfile componentization details
- ACRE algorithm implementation details

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

REAFFIRM Introduction

- REAFFIRM: Reverse Engineer, Analyze, and Fuzz FIRMware
  - Supports wide variety of firmware and software
  - Unpacks, extracts, rehosts, and harnesses
  - Supports testing / fuzzing of firmware on commodity hardware
  - Infers high-level function capabilities (presented at HCSS-2023)
- REAFFIRM is a toolbox
- Reverse Architecting is now being added
  - Binary componentization
  - Design-to-implementation mapping

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Factor: Sequence Graph

Binary	Lang/ISA	Map # Fns	Bin # Fns	SG-CU P	SG-CU R	SG-CU F1	SG-Lib P	SG-Lib R	SG-Lib F1
OpenDPS	C – ARM T32	244	647	82.6%	5.93%	11.1%	88.4%	1.55%	3.05%
UPSat	C – ARM T32	422	578	63.1%	8.41%	15.2%	74.0%	4.02%	7.69%
(propriet.)	C – ARMv5TE	453	510	59.4%	4.87%	9.11%	84.4%	0.08%	0.16%
bbox-x64-dyn	C++ - x64 dynamic linked	2877	3416	59.4%	3.83%	7.30%	74.5%	0.07%	0.14%
bbox-mips	C++ - MIPS32	3744	4193	59.4%	4.87%	9.11%	84.4%	0.08%	0.16%
bbox-x64-stat	C++ - x64 static linked	4330	5661	59.8%	5.11%	9.54%	82.5%	0.08%	0.16%
PX4	C++ - ARM T32	9666	8514	15.5%	0.81%	1.61%	16.6%	0.39%	0.77%
(propriet.)	Ada - PPC	14944	13299	78.7%	0.01%	0.02%	78.7%	0.01%	0.02%

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Factor: Call Graph

Binary	Lang/ISA	Map # Fns	Bin # Fns	CG-CU P	CG-CU R	CG-CU F1	CG-Lib P	CG-Lib R	CG-Lib F1
OpenDPS	C – ARM T32	244	647	4.73%	0.56%	1.09%	5.72%	0.17%	0.33%
UPSat	C – ARM T32	422	578	22.4%	2.11%	3.87%	40.0%	1.53%	2.96%
(propriet.)	C – ARMv5TE	453	510	21.1%	0.97%	1.88%	59.8%	0.71%	1.41%
bbox-x64-dyn	C++ - x64 dynamic linked	2877	3416	23.7%	3.10%	5.73%	61.3%	0.11%	0.23%
bbox-mips	C++ - MIPS32	3744	4193	47.4%	2.76%	5.27%	83.2%	0.05%	0.11%
bbox-x64-stat	C++ - x64 static linked	4330	5661	18.7%	2.83%	4.98%	61.0%	0.10%	0.21%
PX4	C++ - ARM T32	9666	8514	7.3%	0.34%	0.68%	9.9%	0.21%	0.41%
(propriet.)	Ada - PPC	14944	13299	71.6%	0.01%	0.02%	71.6%	0.01%	0.02%

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Factor: Data-Reference Graph

Binary	Lang/ISA	Map # Fns	Bin # Fns	CG-CU P	CG-CU R	CG-CU F1	CG-Lib P	CG-Lib R	CG-Lib F1
OpenDPS	C – ARM T32	244	647	13.6%	3.11%	5.97%	13.8%	0.77%	1.52%
UPSat	C – ARM T32	422	578	38.2%	6.12%	10.6%	61.1%	3.99%	7.49%
(propriet.)	C – ARMv5TE	453	510	60.5%	14.5%	23.6%	78.5%	4.85%	9.16%
bbox-x64-dyn	C++ - x64 dynamic linked	2877	3416	5.86%	26.5%	10.1%	93.6%	6.06%	11.4%
bbox-mips	C++ - MIPS32	3744	4193	1.33%	30.6%	3.03%	58.9%	15.2%	25.1%
bbox-x64-stat	C++ - x64 static linked	4330	5661	5.11%	24.7%	9.11%	86.8%	4.72%	8.98%
PX4	C++ - ARM T32	9666	8514	-	-	-	-	-	-
(propriet.)	Ada - PPC	14944	13299	69.3%	0.11%	0.22%	69.3%	0.11%	0.22%

GRAMMATECH

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

GT-BCD: what didn't work

- Some things tried worked poorly
  - Dropping SG edges between very large functions ("size-limited" adjacency)
  - "Rare instruction" similarity (e.g., uses floating point) is not a strong signal, and is very difficult to compute

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

Componentization: Mapfile algorithm

- Built as a componentization algorithm
  - Gives output in a consistent form to other algorithms
  - Can be applied by user if mapfile is available
- Support for multiple formats (extensible)
  - GCC, LLVM, XLink
- Recovers:
  - Function grouping in object modules
  - Object module grouping in libraries
  - Higher level structure from source directory organization

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

ACRE Algorithm (1)

- Graph preparation (improve similarity)
  - Remove leaf function nodes (not present in $\mathcal{D}$) from I
  - For mapfile CPZN, remove single-function object files
    - $\mathcal{D}$ never says "place `memcpy()` in `memcpy.c`"
    - This is an implementation detail to support link optimization

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

ACRE Algorithm (2)

- Semantic matching of $\mathcal{D}$ to I
  - Information for each graph created when built
    - $\mathcal{D}$: Names (N) and module sizes (MS)
    - I: Names (N), strings (S), capabilities (C), module sizes (MS)
  - Compute confidence levels (*Jaccard-like*)
    - NxN, NxS, NxC, MSxMS
    - NxC is “library aware” – knows what cap’s library represents
  - Weight and combined (mainly, MSxMS lower

© 2024 GrammaTech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A: Approved for public release: distribution is unlimited.

ACRE Algorithm (3)

- Tracing parentage of each $\mathcal{D}$ and $\mathcal{I}$ component
  - Assume our confidence in $\mathcal{D}$ is total (conf=1.0)
  - Assume we are less sure about $\mathcal{I}$ (conf=0.25, *arbitrary magic*)
- Trivial but we need values for later calculation

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

ACRE Algorithm (4)

- Propagate structural relations
- Similiar to "squaring" in Sapper analogy model
  - Given $C_{\mathcal{D}} \cong C_{I}$ & parent($P_{\mathcal{D}}$, $C_{\mathcal{D}}$) & parent(P_{I}, C_{I}) impute $P_{\mathcal{D}} \cong P_{I}$ with product of confidence levels
  - If multiple terms impute same parent "combine"
  - Explored: max, mean, saturating sum, inverse product, geometric mean
  - Not highly sensitive; max works OK and is fast

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

ACRE Algorithm (5)

- Combine semantic and structural confidence
- Separate inputs must be combined
  - Again, multiple possibilities, but max is OK
- Assign confidence = 1.0 for knowns: $\mathrm{Sys}_{\mathcal{D}} = \mathrm{Bin}_{I}$
- Collect final numbers for each *proposed* map
- At this point there are multiple candidates

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.

ACRE Algorithm (6)

- Best-first, implementation-driven map assignment
  - Chose mapping with highest score
  - Add this mapping to final output
  - Remove from pool all mappings with this I component
    - Disallowing N:1 mappings
  - Downgrade all mappings with this $\mathcal{D}$ component
    - 1:N mappings possible, but 1:1 is preferred
    - Uses magic value 0.9: “1:2 maps are 90% as likely as 1:1 maps”

© 2024 Grammatech, Inc. This work is licensed under CC BY-NC-ND BY-NC-ND
Distribution Statement A. Approved for public release: distribution is unlimited.