Arbitrary Precision and Complexity Tradeoffs for Gate-Level Information Flow Tracking

Andrew Becker[†], Wei Hu[‡], Yu Tai[§], Philip Brisk[‖], Ryan Kastner[‡] and Paolo Ienne[†]
[†]Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
[‡]University of California, San Diego, La Jolla, CA 92093
[§]Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China
[‖]University of California, Riverside, Riverside, CA 92521
{andrew.becker, paolo.ienne}@epfl.ch; {weh040, kastner}@ucsd.edu;
taiyu@mail.nwpu.edu.cn; philip@cs.ucr.edu

ABSTRACT

Hardware has become an increasingly attractive target for attackers, yet we still largely lack tools that enable us to analyze large designs for security flaws. Information flow tracking (IFT) models provide an approach to verifying a hardware design's adherence to security properties related to isolation and reachability.

However, existing precise IFT models are usually too complex to actually use. Queries may fail to finish even for small designs when verifying relatively simple properties. It is possible to create less complex models, but these come at the cost of a severe loss of precision—they frequently indicate a property fails when in fact it passes, which means verification requires extensive additional manual investigation.

We present a new method to bridge the chasm between precision and complexity in a finer-grained, controlled, and disciplined manner. Our method allows using the most appropriate precision/complexity tradeoff for the design size and available computing resources, meaning it is now possible to create models that are not too complex to be usable, but which offer more precision (fewer false positives) than was previously possible.

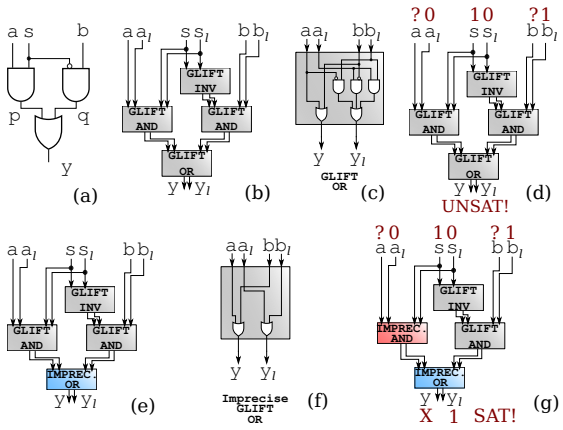

Figure 1: (a) Original design netlist, (b) Corresponding GLIFT "constructive method" model, (c) Whiteboxing a GLIFT `OR` cell, (d) Illustration of a query on a GLIFT model, (e) Precisely simplified GLIFT model, (f) Imprecise "all-`OR`" GLIFT cell, (g) Imprecisely simplified GLIFT model.

1. INTRODUCTION

The constant increase in semiconductor hardware design complexity practically ensures that modern chips will contain security flaws. Typical supply chains are opaque [4], and methods for sabotage are so stealthy [2], that malicious design modifications could remain undetected for years. Furthermore, attacks exploiting hardware design flaws are increasingly common, and the target scope includes everything from personal mobile devices to national air defence radar systems [1].

Automated analysis methods that can verify a system adheres to high-level security specifications could eliminate the possibility of certain exploitable flaws. Information Flow Tracking (IFT) models [3, 11], for example, can help to verify non-interference [6], or that if one assigns 'high' and 'low' security labels to inputs and outputs, 'low'-labelled outputs are functionally unaffected by 'high'-labelled inputs. This means IFT models can be used to check properties like isolation and reachability [6], useful e.g. to detect hardware Trojans [4]. Of particular interest are gate level IFT (GLIFT) models [10] which add a "taint" label to each signal in the raw design netlist and model how tainted information can flow gate-to-gate from inputs to outputs.

Fig. 1(a) shows a simple logic circuit (a `MUX`) and Fig. 1(b) shows how the typical constructive mapping approach replaces each gate in the netlist with a GLIFT cell that implements the very same logic funtionality but also includes extra inputs and outputs to label and track tainted flows. Fig. 1(c) details the GLIFT `OR` cell, which expresses that the output is tainted either if both inputs are tainted or if one input is tainted and the other is at the logic value `0` (that is, it exposes the state of the tainted input to the output). A typical Boolean SAT query on this GLIFT model, albeit a trivial one in this elementary example, is shown in Fig. 1(d): the user asks whether there is any condition under which the tainted input `b` may leak to the output given the knowledge

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

DAC '17 June 18-22, 2017, Austin, TX, USA

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4927-7/17/06.
DOI: http://dx.doi.org/10.1145/3061639.3062203

of some of the inputs and labels; the answer is clearly "no" in this case (and hence the query is UNSAT) because $s = 1$ and the output is logically connected to `a` (remember that the circuit is a `MUX`).

Fig. 1(e) shows the result of a recent precise simplification approach [7] where the GLIFT `OR` cell at the circuit output is replaced with a lower-cost imprecise version (a GLIFT `XOR` cell) shown in Fig. 1(f). The simplification is possible because the internal nodes `p` and `q` are mutually exclusive (due to the signal `s`) and thus the output GLIFT `OR` cell can be replaced with a GLIFT `XOR` gate without any change in the functionality—even though taken alone, a GLIFT `XOR` cell overapproximates label propagation of a GLIFT `OR` cell. Thus, the model is smaller but the propagated label value is still perfectly correct under all conditions.

Finally, notice that in Fig. 1(g) the same simplification in the taint propagation logic is applied to one of the GLIFT `AND` cells. Again, the simpler imprecise cell overapproximates the label (is `1` when it should be `0`), but in this case the label may propagate to one of the model's label outputs. This GLIFT model is now imprecise because that same SAT query is now satisfiable: the GLIFT model now reports a *false positive*: a reported information flow that does not actually exist. It is worth emphasizing that this does not compromise security in any way; it burdens the designer to assess whether each reported flow is indeed real.

Unfortunately, SAT queries on precise GLIFT models can take weeks, or even longer, for moderate- to large-sized designs. Designers are thus typically forced to use an "all `OR`" model, where this simplification (replacing the correct label propagation logic with a simple `OR` gate) is applied everywhere. This makes it possible to query models of more complex designs, but there is no control whatsoever on introduced false positive flows.

Contribution: This paper describes automation techniques for circuit designers to make principled tradeoffs between the precision and complexity of GLIFT models. Specifically, we propose two new QBF-SAT problem formulations and describe algorithmically how to apply the solver to iteratively generate a simplified GLIFT model for a target level of imprecision. Our first formulation allows imprecision by forcing the solver to choose a set of bit vectors to exclude from the precise equivalence constraint; this approach works in principle, but suffers from severe scalability issues. Our second formulation addresses this shortcoming by instead forcing the solver to choose a set of bit vector *patterns*, including don't-cares, subject to additional constraints on the allowed number of extra imprecise flows. We analyze the tradeoff space for an example set of designs and show how our approach can be used to create GLIFT models that trade-off between precision and complexity in a reasonably controllable way.

2. RELATED WORK

Precision and complexity are contradictory modelling goals [9]. In practice, formally verifying properties of GLIFT models is intractable without accepting very low precision. Designers may simply abandon GLIFT in favor of a higher-level model [11], may try to simplify without sacrificing precision [7], or may use very imprecise models with "all `OR`" label propagation [3].

Constructively-generated GLIFT logic often exhibits internal redundancy [8, 9] which can be exploited to generate simplified GLIFT models that do not incur reduced precision [7]; however, the internal redundancy varies from design to design, and the simplifications that can be achieved in practice are limited. Alternatively, applying constructive mapping after first applying some logic synthesis optimizations to the design allows generating slightly simpler GLIFT models, at the cost of some precision [7]. However, the optimization achieved is limited, not controllable, and does not adapt to verification needs.

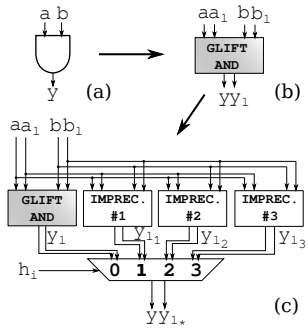

Figure 2: Original circuit gate (a), to its corresponding GLIFT cell (b), to a GLIFT supercell (c). h_i tunes the (im)precision of the supercell.

In response, this paper introduces a novel methodology to make arbitrary tradeoffs between GLIFT model complexity (correlated to GLIFT query time [7]) and precision, enabling tractable GLIFT model queries that sacrifice considerably less precision than with "all `OR`".

3. PRECISE GLIFT MODELS

3.1 Instrumented Model Construction

Fig. 2(a) shows one gate in the original design netlist. Fig. 2(b) shows the original, locally-precise GLIFT cell after constructive mapping. Precisely simplifying the model requires constructing a so-called "instrumented model", incorporating it in a QBF-SAT problem formulation, and iteratively driving the solver to find the best solution. The instrumented model is constructed by replacing each GLIFT cell with a GLIFT "supercell" (Fig. 2(c)), where a multiplexer selects a label propagation function according to a select line `hi` whose value will eventually be determined by the solver. The choice of propagation functions determines the generated model's precision and complexity.

While so far we've only discussed `OR` gates as imprecise label propagation functions, others are possible, and each with varying costs. By allowing a choice of alternative functions, the solver has potentially more freedom to choose alternative functions elsewhere without changing overall model functionality. Thus, although the choice of propagation function is a local substitution, it affects the global context.

The eventual solution specifies a concrete value for each `hi` in the instrumented model, a two-bit signal that selects the appropriate propagation function, denoted y_{l_*}. Once the selection has been made, the instrumented model is converted to RTL and each `hi` is replaced with its constant value, eliminating the unselected GLIFT cells and the multiplexer, leaving only the one GLIFT cell with the solver-chosen propagation function for each original gate.

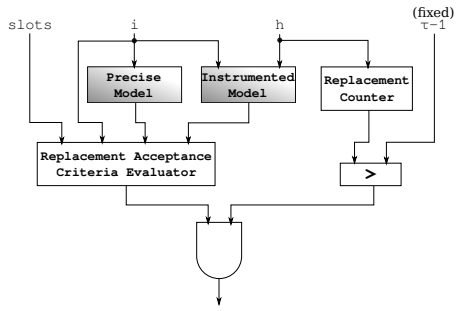

Figure 3: Diagram of the QBF-SAT problem for the explicit method. The QBF-SAT solver tries to force the output of this circuit to `1` for all possible values of the primary inputs `i` under the constraint that the configuration found must use at least τ imprecise GLIFT cells. The solution of the QBF-SAT problem returns an assignment for `h` (the chosen configurations for the supercells) and `slots` that limits global model imprecision. That limit is effected by the Replacement Acceptance Criteria Evaluator, whose operation is visualized in Fig. 4.

3.2 SAT and QBF-SAT

This approach is based on an extension to the following variant of the Boolean Satisfiability (SAT) problem, which can determine the universal truth value of a Boolean formula:

$$\forall i \in \{0,1\}^n : \phi(i) \tag{1}$$

In other words, SAT solvers can determine if some acceptance function ϕ evaluates to `1` under all possible values of `i`.

Quantified Boolean Formula Satisfiability (QBF-SAT) adds an extra layer of quantification:

$$\exists h \in \{0,1\}^m . \forall i \in \{0,1\}^n : \phi(h,i) \tag{2}$$

Thus, QBF-SAT can be used to determine if there exists *any* value for `h` for which $\phi(h,i)$ evaluates to `1` under all possible values of `i`. A QBF-SAT solver lies at the core of both precise and imprecise GLIFT model synthesis.

3.3 Simplification as QBF-SAT

The QBF-SAT problem is formulated so the acceptance function $\phi(h,i)$'s output is `1` when both of the following conditions are satisfied:

- when `h`–the concatenation of all supercells' select lines–configures at least some minimum number τ of supercells to use a locally-imprecise propagation function
- *and* when the instrumented model's output label values are identical to those of the original precise GLIFT model

By iteratively solving and adjusting τ, the GLIFT model with the most possible replacements can be generated.

4. IMPRECISE GLIFT MODELS

Precise GLIFT simplification, as described in the preceding section, exploits internal redundancy created by the constructive method to simplify some GLIFT cells without affecting model precision. However, the strict equivalence constraint still yields costly implementations. In this section, we relax the strict equivalence constraint, which exposes more simplification opportunities as long as we are willing to accept some 'false-positive' detected flows in addition to those that are already present in the constructive GLIFT model. In other words, an imprecise GLIFT model may report a flow (i.e., an output with label `1`) when the precise model reports that no such flow exists (i.e., the same output is labeled `0`). We describe techniques to formulate the QBF-SAT constraints to allow false positives for specific sets of input vectors, where the solver automatically chooses the best such vectors (it is also possible to restrict false positives to only a subset of output labels). By changing one parameter to these constraints, the user can explore tradeoffs between GLIFT model precision and complexity.

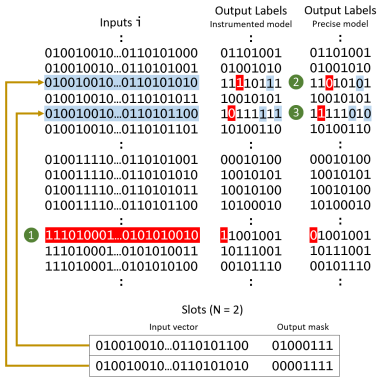

Figure 4: The imprecision acceptance criteria for the explicit method. This is an example of an invalid supercell configuration, showing how the slots determine which imprecision-generating inputs are accepted. Here the instrumented model is not a valid approximation due to three problems (indicated in negative, red): (1) The instrumented model's outputs change under three input vectors but there are only two slots. (2) When the first input vector is applied, one false positive is not in a position allowed by the slot's output mask. (3) With the second input vector, one of the changes is not a false positive, but a false negative.

4.1 Explicit Acceptance by Bit-Vectors

Let `(x)` denote a bit-vector of length |`x`| whose binary encoding is equal to `x`. A slot is a bit-vector `(s)` containing an "input vector" and an "output mask", and describes which model input values may trigger a false positive, and in which output label(s). When generating an imprecise GLIFT model, the designer specifies a number of `N` slots, thereby providing a degree of control over the amount of imprecision that may be inserted.

Fig. 3 illustrates the QBF-SAT problem instance that is used to simplify a precise GLIFT model to an imprecise one. The user provides a precise GLIFT model, which is automatically used to create an instrumented model (in which GLIFT gate replacement configuration options are available, as in Fig. 2(c)). The Replacement Acceptance Criteria Evaluation function ensures that the constraints encoded in the slots (whose contents are themselves determined by the solver) are checked. For input bit vectors that don't match a slot, the imprecise and precise GLIFT models must produce identical labels; in case of a match, the labels may be identical or a false positive. The user also provides an in-

teger parameter, τ, which is a lower bound on the number of locally-imprecise GLIFT cell replacements to be made; the Replacement Counter and comparator ($>$) ensure that this lower bound is achieved. The `AND` gate at the bottom ensures that both criteria are satisfied, i.e.: (1) the imprecise GLIFT model accounts properly accounts for all $2^{|i|}$ slot and non-slot bit vectors; and (2) at least τ locally-imprecise GLIFT cells are used instead of precise GLIFT cells.

4.1.1 *Imprecision Acceptance Criteria*

Fig. 4 illustrates several relevant aspects of the imprecise acceptance criteria. In this example, there are $N = 2$ slots, each of which has a corresponding output mask. For each bit in the output mask, a value of `1` indicates that a false positive is allowed in that bit position, and a value of `0` indicates that a false positive is not allowed there.

Three errors occur in Fig. 4 that would not be accepted by our model: First, a bit vector not matching any slot yields a false positive. Second, a bit vector that *does* match a slot yields a false positive corresponding to a bit position whose output mask has a value of `0`. Third, the model generates a false negative in an allowable bit position, rather than a false positive. Our problem formulation ensures that these types of errors do not occur.

4.1.2 *QBF-SAT Formulation*

The added elements of imprecision require an updated QBF-SAT problem formulation that goes beyond the precise formulation introduced in Section 3.2. The formulation for the function illustrated in Fig. 3 is:

$$\exists(h, slots) \in \{0,1\}^m . \forall i \in \{0,1\}^n : \phi(h, slots, i) \qquad (3)$$

Notice that the function ϕ in Equation 3 has no parameter τ, which is shown as an input in Fig. 3. We fix the value of τ for each QBF-SAT problem instance; we iteratively adjust τ and re-solve, using a binary search method to generate a sequence of models with progressively more replacements, quickly converging on the model that maximizes the number of replaced GLIFT cells while still adhering to the constraints described above.

4.1.3 *Solver Runtime*

Fig. 5 reports the runtime of the QBF-SAT solver as a function of the number of slots (`N`) provided by the user, given a one-hour time limit. For $N \leq 56$, the solver was able to replace seven cells in 207 seconds or less; for $N = 64$, the solver could only find five replacements within the allotted hour, so a dashed line is shown to the timeout. These results indicate that acceptance by bit vectors scales poorly.

4.2 Acceptance by Patterns

Our solution is to calculate acceptance not by bit vectors, but by *patterns* of bit vectors. Acceptance by patterns allows each slot to encode allowable false positive flows for multiple bit vectors. A pattern includes one or more don't-care values encoded by an `X` in place of an individual bit, as shown in Fig. 6; a pattern with `j` don't-cares covers 2^j distinct bit vectors.

Fig. 7 depicts a new acceptance function. It is similar to the function shown in Fig. 3, but with a few key differences. The slot encoding allows for don't-care bits (not shown); the user specifies a parameter `MaxFP` which provides an upper limit on the number of unique bit vectors that can be

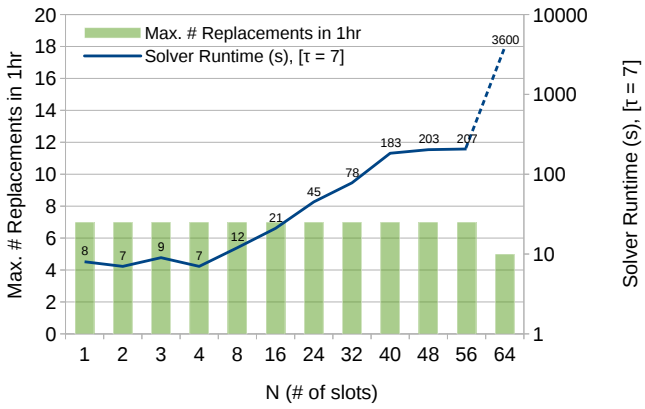

Figure 5: A chart showing with columns (axis on left) the maximum number of GLIFT cell replacements and the QBF-SAT solver runtime to find those replacements for the `too_large` experiment with various `N` values (i.e. numbers of slots), given a 1-hour solver timeout. The data at `N=64` shows that the solver was not even able to find the same seven replacements it could with fewer slots; it could only find five. Note how the solver runtime increases rapidly, and yet we find no additional GLIFT cell replacements.

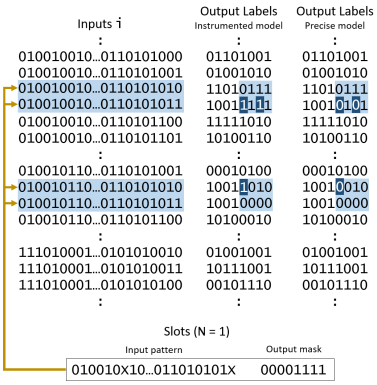

Figure 6: The imprecision acceptance criteria for the pattern method. This figure shows how changes to the GLIFT model truth table (i.e. false positives) are allowed in the pattern method and how the estimated upper bound on additional false positives is computed. Individual input vectors are now replaced by patterns including don't-cares. Acceptable false positives (three in the example, indicated in negative, dark blue) must match at least one of the input patterns' covered rows and also in the output mask's columns.

covered by patterns (note that multiple patterns may cover the same bit vector); and a "Bound Evaluator" component, which enforces the aforementioned upper bound. This provides the user an extra degree of freedom in addition to the number of slots. Specifically, `MaxFP` can be interpreted as an upper bound on the number of bit vectors that may introduce at least one false positive flow.

The truth table cell coverage for a given slot is computed as: $2^{\texttt{HW(input\_mask)}} \cdot \texttt{HW(output\_mask)}$ where `HW` is the Hamming weight function. The coverage value for each slot is then summed and compared to the provided threshold `MaxFP`.

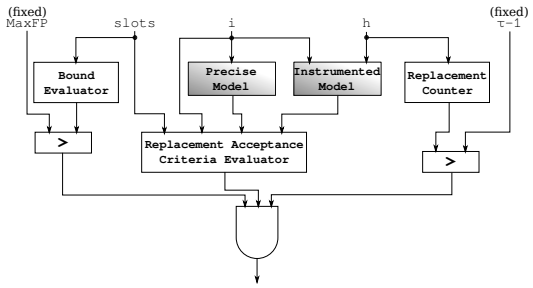

Figure 7: The QBF-SAT problem for the patterns method. The QBF solver addresses essentially the same problem as the explicit method with two important differences: (1) The semantics of the slots are now changed to contain don't-care patterns, and the acceptance function is changed accordingly. (2) The upper bound on how imprecise the `h` can validly make the model is now limited by the constant parameter `MaxFP`. The criteria used in this circuit are illustrated in Fig. 6.

The appropriate parameter value for `MaxFP` varies from design to design, and some sense of the number of *existing* flows in the design can help to bound the desired number of false-positive flows. Our approach is to simulate the original precise model with a relatively small number (2^{20}) of uniformly random input and input label values. We then scale the result to the size of the model input space $2^{|I|}$, where $|I|$ is the number of inputs and input labels. This provides a rough estimate of the number of original flows, allowing for quick calibration of the `MaxFP` parameter.

This is not a hard upper bound, and could easily overestimate the number of precise flows when scaled to the size of the full model input space. As an example, the `alu4` benchmark circuit has 28 model inputs, so its `MaxFP` parameter is 256 times the number of flows sampled in the precise model.

5. EXPERIMENTAL RESULTS

5.1 Methodology

To empirically verify our claim that we can generate circuits with arbitrary tradeoffs between added false positives and complexity, we must have a method to measure at least an estimate of the actual number of false-positive flows that a given imprecise model produces. To estimate, we use a set of designs from the standard IWLS benchmark set, and for each we use the same pseudorandomly generated 2^{20} model input vectors used for estimating `MaxFP`, which were generated using Linear Feedback Shift Registers (LFSRs) with periods longer than 2^{20}. Then we simply count the number of flows detected and subtract the number of flows detected for the same sample with the precise model.

Due to the time and expense of an exhaustive exploration of the possible configuration space for each experiment, we employed a binary search method to find the maximum possible number of replacements given `N` (the number of slots) and `MaxFP`, described by the pseudocode in Fig. 8, and ran multiple experiments changing `MaxFP` to estimate 80%, 60%, 40%, 20%, 10%, 5%, 2%, and 1% false positive rates. All experiments ran on Xeon E5-2680 v3 processors with at least 64GiB of available RAM, with a QBF-SAT instance timeout of 1 hour, using Yices 2.5.1 [5] in "exists-forall" mode.

```
delta := infinity
tau := None
N := //...
MaxFP := //...
solution = Solve(tau, N, MaxFP)
min_fail := 0

if not solution:
        exit
while delta > 1:
        delta := (min_fail - max_succ)
        tau := max_succ + (delta / 2)
        solution := Solve(tau, N, MaxFP)
        if solution:
                max_succ := size(get_replacements(solution))
        else:
                min_fail := tau
```

Figure 8: Pseudocode for the configuration space exploration algorithm. Given `N` false positive pattern slots and a bound on the percent of false-positives in the generated model, this algorithm finds the configuration for the generated GLIFT model with the maximum number of replacements possible given the values for `N` and `MaxFP`.

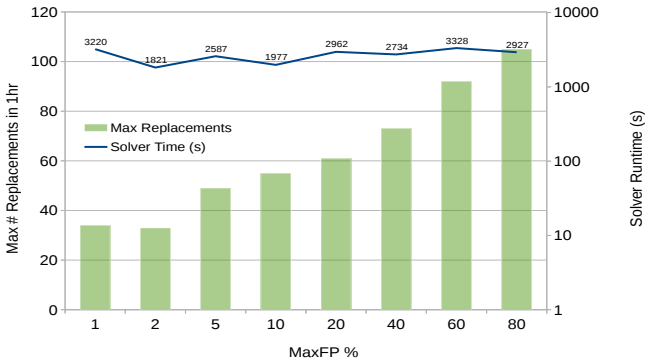

Figure 9: Similar to Fig. 5, but with `MaxFP` on the horizontal axis as the user-controlled variable, with two pattern slots, for `too_large`. Note how many more replacements (columns, left axis) are found than with the 'explicit' method, and the runtime (line, right axis) stability.

5.2 Discussion

Fig. 9 serves as a counterpoint to Fig. 5. With the 'pattern' technique, we can find many times the number of replacements as 'explicit', and with more reasonable runtime, too. Some experiment instances used almost the entire alloted hour of solver time while others finished within minutes. Space constraints prevent us from presenting all the runtime data; this example serves as a representative indication of what to expect.

In Fig. 10 we show the actual area reduction achieved versus the measured false positive rate. The results reported here are only for the results of the search algorithm in Fig. 8 with two pattern slots, not intermediate steps. While this is still a busy chart, one can see how generally a higher measured false positive rate corresponds with a bigger area reduction. One can also see that the region between the "all-`OR`" data points is not quite covered. This is likely due to our approach maximizing the *number* of imprecise GLIFT cells, not the *simplicity* of imprecise GLIFT cells. We also suspect the visible "noise" in the `ttt2` data is due to the same cause. Future work will explore, for example, weighting each

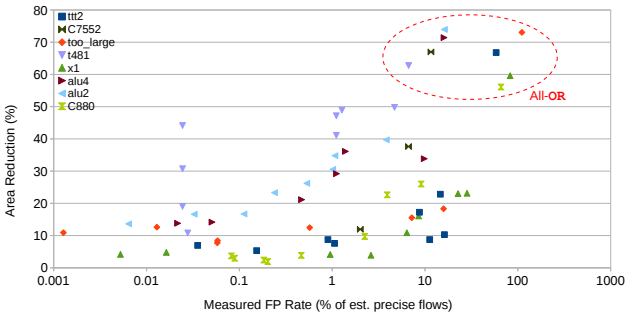

Figure 10: A comparison of area reduction to the measured additional false positive rate (as a percentage of the original number of flows) among the sampled 2^{20} input vectors, given two pattern slots. Allowing extra false positives reduces the model's area, and we can generate models with arbitrary imprecision. The corresponding "all-OR" models are highlighted.

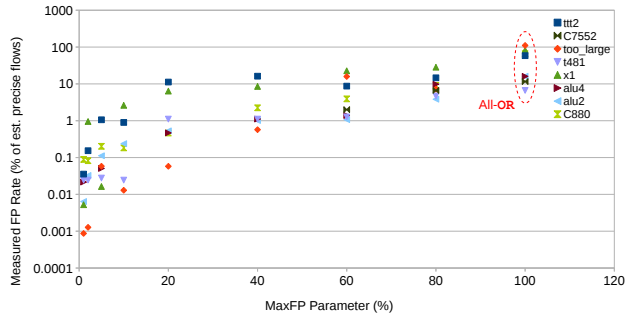

Figure 11: The measured additional false positive rate (on a log axis) among the sampled 2^{20} input vectors versus the `MaxFP` parameter. The `MaxFP` bound is loose, but clearly effective at controlling the actual false positive rate. The corresponding "all-OR" models are again highlighted.

supercell option; this weight could be the number of gates in the chosen propagation function to potentially make a better proxy for simplicity. Still, overall, these data show that we do effectively trade off complexity (by proxy of area) and the false positive rate for the generated GLIFT models.

In Fig. 11, we show the measured additional false positive rates for the same experiments versus the `MaxFP` parameter used. Here one can clearly make out that increasing the bounded false positive rate generally induces more aggressive imprecision.

Together, Fig. 10 and Fig. 11 show that not only can we trade off complexity and imprecision, but we have a controllable and flexible method to do so, as well.

6. CONCLUSIONS

Gate-level information flow tracking offers the promise of verifying important security properties at the Boolean gate level. Unfortunately, precise GLIFT models are often too complex to practicably use for verifying security properties. Previous work mostly involved extreme simplifications like reducing all GLIFT label propagation to `OR`; more recent work introduced some limited means of trading a small amount of precision for a reduction in complexity, but without any controllability. While imprecision does not reduce security, it adds the burden of manually verifying all reported flows. Excessively imprecise models are of limited use, however, as the false positive rate is very high. We present the first known method to systematically generate imprecise GLIFT models with a controllable tradeoff between precision and complexity, potentially allowing the use of more precise models than previously was possible and reducing manual verification burden.

In future studies we hope to reduce the complexity of the false positive rate calculation logic while sacrificing as little controllability as possible, to assign weights to alternative propagation functions, and to demonstrate how imprecise GLIFT models can help with verification in the real world.

7. REFERENCES

[1] S. Adee. The hunt for the kill switch. *Spectrum, IEEE*, 45(5):34–39, May 2008.

[2] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans. In *the 15th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'13, pages 197–214, Berlin, Heidelberg, 2013. Springer-Verlag.

[3] M. Bidmeshki and Y. Makris. Vericoq: A verilog-to-coq converter for proof-carrying hardware automation. In *2015 IEEE International Symposium on Circuits and Systems, ISCAS 2015, Lisbon, Portugal, May 24-27, 2015*, pages 29–32, 2015.

[4] G. Bloom, E. Leontie, B. Narahari, and R. Simha. Hardware and security: Vulnerabilities and solutions, 2012.

[5] B. Dutertre. Yices 2.2. In A. Biere and R. Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[6] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, 1982.

[7] W. Hu, A. Becker, A. Ardeshiricham, Y. Tai, P. Ienne, D. Mu, and R. Kastner. Imprecise security: Quality and complexity tradeoffs for hardware information flow tracking. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 95:1–95:8, New York, NY, USA, 2016. ACM.

[8] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. Theoretical fundamentals of gate level information flow tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(8):1128–1140, Aug 2011.

[9] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. On the complexity of generating gate level information flow tracking logic. *IEEE Transactions on Information Forensics and Security*, 7(3):1067–1080, June 2012.

[10] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood. Complete information flow tracking from the gates up. In *international conference on Architectural support for programming languages and operating systems*, ASPLOS'09, pages 109–120, New York, NY, USA, 2009.

[11] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers. A hardware design language for timing-sensitive information-flow security. In *the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 503–516, New York, NY, USA, 2015.