IX. Appendix

A. Algorithm for Adapter Synthesis

Here we present the pseudocode for our adapter synthesis algorithm, which was summarized in Figure 2. Algorithm 1 presents the main adapter synthesis loop, Algorithm 2 presents the `CheckAdapter` procedure that generates counterexamples, and Algorithm 3 presents the the `SynthesizeAdapter` procedure that generates candidate adapters. `CheckAdapter` and `SynthesizeAdapter` are both implemented as calls to a symbolic executor.

Input : Target code region T, reference function R, and adapter family $\mathcal{F}_{\mathcal{A}}$
Output: (input adapter $\mathcal{A}_{in}$, output adapter $\mathcal{A}_{out}$) or *null*
$\mathcal{A}_{in} \leftarrow$ default-input-adapter;
$\mathcal{A}_{out} \leftarrow$ default-output-adapter;
test-list $\leftarrow$empty-list;
while *true* **do**
    counterexample $\leftarrow$ `CheckAdapter` ($\mathcal{A}_{in}$, $\mathcal{A}_{out}$);
    if counterexample *is null* **then**
        return $(\mathcal{A}_{in}, \mathcal{A}_{out})$*;*
    else
        test-list.append(counterexample);
    end
    $(\mathcal{A}_{in}, \mathcal{A}_{out}) \leftarrow$ `SynthesizeAdapter` (test-list);
    if $\mathcal{A}_{in}$ *is null* **then**
        return *null;*
    end
end

Algorithm 1: Counterexample-guided adapter synthesis

Input : Concrete input adapter $\mathcal{A}_{in}$ and output adapter $\mathcal{A}_{out}$
Output: Counterexample to the given adapters or *null*
args $\leftarrow$ symbolic;
while *execution path available* **do**
    target-output $\leftarrow$ execute T with input args;
    reference-output $\leftarrow$ execute R with input *adapt(*$\mathcal{A}_{in}$*,* args*);*
    if *! equivalent(*target-output*, adapt(*$\mathcal{A}_{out}$*,* reference-output*))* **then**
        return *concretize(*args*);*
    end
end
return *null;*

Algorithm 2: `CheckAdapter` procedure used by Algorithm 1. T and R are as defined in Algorithm 1.

Input : List of previously generated counterexamples test-list
Output: (input adapter $\mathcal{A}_{in}$, output adapters $\mathcal{A}_{out}$) or *null*
$\mathcal{A}_{in} \leftarrow$ symbolic input adapter;
$\mathcal{A}_{out} \leftarrow$ symbolic output adapter;
while *execution path available* **do**
    eq-counter $\leftarrow 0$;
    while eq-counter $<$ *length(*test-list*)* **do**
        target-output $\leftarrow$ execute T with input test;
        reference-output $\leftarrow$ execute R with input adapt($\mathcal{A}_{in}$, test));
        if *equivalent(*target-output*, adapt(*$\mathcal{A}_{out}$*,* reference-output*))* **then**
            eq-counter $\leftarrow$ eq-counter + 1;
        else
            break;
        end
    end
    if eq-counter == *length(*test-list*)* **then**
        return *(concretize(*$\mathcal{A}_{in}$*), concretize(*$\mathcal{A}_{out}$*));*
    end
end
return *null;*

Algorithm 3: `SynthesizeAdapter` procedure used by Algorithm 1. T and R are as defined in Algorithm 1. The form of the resulting adapters ($\mathcal{A}_{in}$, $\mathcal{A}_{out}$) is dictated by $\mathcal{F}_{\mathcal{A}}$.

B. Type Conversion Adapter Encoding

In Section III-C we showed how an argument substitution adapter can be encoded using symbolic variables and a Vine IR formula. Here we show how to encode a type conversion adapter that extends the one from Section III-C to allow sign extension from the low 16 bits of a value:

```
y_type == 1 ? y_val :
  ( y_type == 0 ?
    ( y_val == 0 ? x1 :
      ( y_val == 1 ? x2 : x3 ))
  :
    cast( cast(
      ( y_val == 0 ? x1 :
        ( y_val == 1 ? x2 : x3 ))
      L ) S )
```

This Vine IR expression begins in the same way as the one shown in Section III-C, setting y to be the constant value *y_val* if *y_type* is 1. If *y_type* is 0, the expression performs argument substitution based on the value in *y_val*. If *y_type* is any value other than 0 or 1, it sign extends the low 16 bits in the target input specified by *y_val* to the bitwidth of the reference function argument. The sign extension and low-16-bits-extraction operations are denoted by `S` and `L` respectively in Listing IX-B. In general, the only work required to add support to our tool for a new adapter family is to design a way of encoding elements of that family using symbolic variables, constraints on the values of those variables, and a Vine IR expression dictating how to interpret the values of those variables.

C. Adapter Synthesis Running Times

We present histograms of the total runtimes required to find adapters for the `clamp` and `median` reference functions in Figures 6 and 7. As in Figure 5, most adapters were found before 300 seconds.

D. Reverse Engineering Expanded Tables

Here we present the full versions of the table presented in Section IV-B. Table III shows results for the cases where an adapter was found, Table IV shows results for the cases where the target region and reference function were deemed unsubstitutable, and Table V shows results for the cases where there was a timeout. The column labelings are generally as in Table I. `AS` stands for adapter synthesis and `CE` stands for counterexample generation. The `#N` within parenthesis after each reference function name indicates the number of arguments taken by that reference function.

TABLE III: Metrics for adapters for all reference functions

function name	#	#full	#clusters	steps	total time (solver)	CE total time (solver)	CE last time (solver)	AS total time (solver)	AS last time (solver)
clamp	683	177	110	12.903	99.272 (12.099)	17.110 (0.941)	1.880 (0.282)	82.163 (11.158)	32.490 (4.253)
prev_pow_2(#1)	32	0	6	4.688	6.125 (0.266)	4.312 (0.144)	0.875 (0.053)	1.812 (0.122)	0.938 (0.063)
abs_diff(#2)	575	5	75	10.517	19.981 (1.331)	12.944 (0.487)	1.120 (0.095)	7.037 (0.844)	1.843 (0.276)
bswap32(#1)	115	8	19	8.67	16.565 (1.235)	12.313 (0.984)	1.000 (0.227)	4.252 (0.251)	1.226 (0.089)
integer_cmp(#2)	93	5	15	9.645	21.419 (2.246)	8.839 (0.598)	1.280 (0.275)	12.581 (1.648)	4.742 (0.630)
even(#1)	3	2	3	5.667	11.333 (0.558)	7.000 (0.312)	2.333 (0.218)	4.333 (0.246)	2.333 (0.154)
div255(#1)	4	0	2	5	6.500 (0.262)	4.000 (0.143)	0.750 (0.051)	2.500 (0.119)	1.500 (0.068)
reverse_bits(#1)	276	0	11	8.978	25.264 (2.926)	16.192 (0.678)	1.978 (0.112)	9.072 (2.248)	1.895 (0.454)
binary_log(#1)	48	0	5	6.708	23.562 (5.870)	10.938 (2.191)	2.125 (0.728)	12.625 (3.679)	8.750 (3.235)
median(#3)	332	42	60	13.669	119.226 (26.739)	17.789 (1.323)	2.250 (0.454)	101.437 (25.416)	33.931 (8.548)
hex_value(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
get_descriptor_ length_24b(#1)	22	9	2	9	16.682 (0.583)	11.909 (0.328)	1.136 (0.091)	4.773 (0.255)	1.591 (0.098)
tile_pos(#4)	5617	407	909	10.902	53.478 (23.124)	10.968 (1.767)	2.836 (1.409)	42.510 (21.357)	18.090 (10.019)
dirac_picture_n_ before_m(#2)	330	2	18	13.224	25.736 (2.974)	13.048 (0.638)	0.855 (0.084)	12.688 (2.335)	2.124 (0.386)
ps_id_to_tk(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
leading_zero_count(#1)	41	0	7	18.561	39.000 (4.529)	22.780 (1.174)	1.000 (0.146)	16.220 (3.355)	2.488 (0.721)
trailing_zero_count(#1)	46	0	4	5.87	16.196 (3.832)	9.109 (1.097)	2.065 (0.738)	7.087 (2.735)	3.478 (1.322)
popcnt_32(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
parity(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
dv_audio_12_to_16(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
is_power_2(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
RenderRGB(#3)	763	2	64	10.814	27.469 (1.518)	17.021 (0.814)	1.046 (0.143)	10.448 (0.704)	2.819 (0.221)
decode_BCD(#1)	0	0	0	0	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
mpga_get_ frame_samples(#1)	22	15	4	5	7.909 (0.887)	5.273 (0.505)	1.182 (0.361)	2.636 (0.381)	1.773 (0.345)

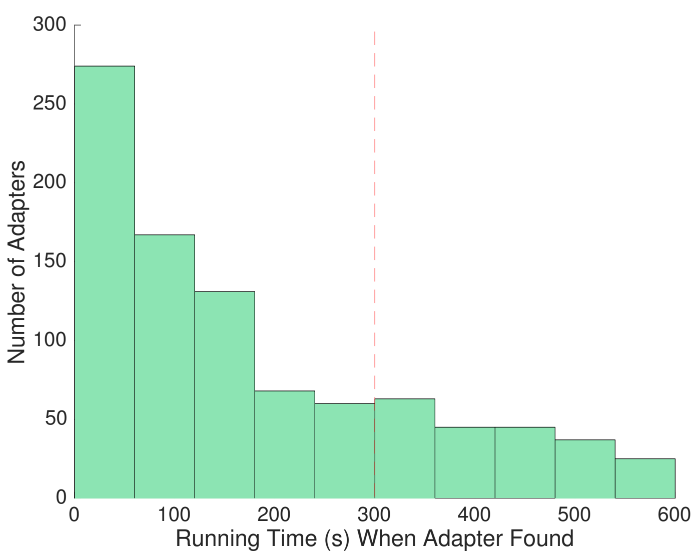

Fig. 6: Time required to synthesize adapters using the `clamp` reference function

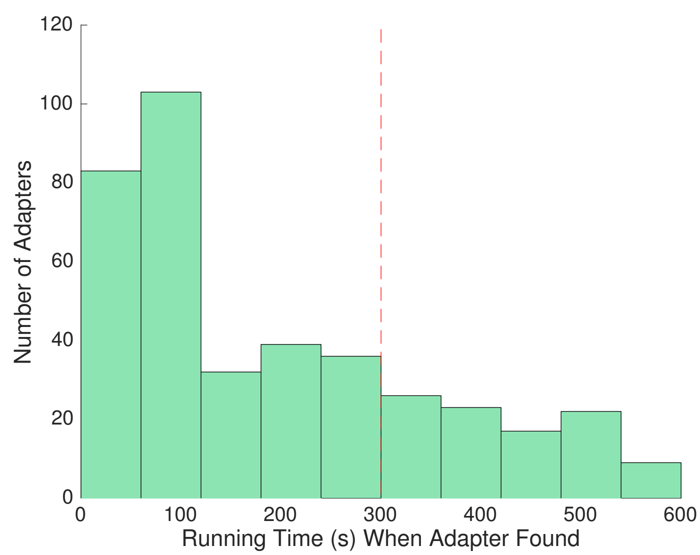

Fig. 7: Time required to synthesize adapters using the `median` reference function

TABLE IV: Metrics for the insubstitutable conclusion for all reference functions

function name	#	steps	total time (solver)	CE total time (solver)	CE last time (solver)	AS total time (solver)	AS last time (solver)
clamp	40553	7.711	63.015 (6.361)	8.171 (0.375)	1.703 (0.112)	54.844 (5.986)	38.464 (4.032)
prev_pow_2(#1)	46767	4.258	7.521 (0.492)	4.833 (0.225)	2.008 (0.154)	2.687 (0.267)	1.502 (0.201)
abs_diff(#2)	46250	8.205	18.735 (1.384)	11.281 (0.411)	2.281 (0.124)	7.453 (0.973)	3.268 (0.562)
bswap32(#1)	46708	4.682	8.184 (0.493)	5.136 (0.196)	1.764 (0.102)	3.048 (0.297)	1.620 (0.217)
integer_cmp(#2)	46467	5.249	15.324 (1.772)	7.850 (0.404)	2.816 (0.177)	7.474 (1.369)	4.640 (0.999)
even(#1)	46823	4.218	12.699 (0.859)	7.088 (0.229)	2.883 (0.149)	5.611 (0.630)	3.881 (0.529)
div255(#1)	46823	4.381	7.568 (0.463)	4.849 (0.206)	1.824 (0.117)	2.719 (0.257)	1.499 (0.196)
reverse_bits(#1)	46541	12.536	50.866 (5.645)	22.051 (0.784)	2.359 (0.103)	28.815 (4.861)	12.573 (1.454)
binary_log(#1)	46528	4.024	25.631 (6.368)	4.848 (0.551)	2.004 (0.136)	20.783 (5.817)	15.253 (4.314)
median(#3)	32171	6.484	89.779 (15.126)	6.598 (0.312)	1.723 (0.097)	83.181 (14.815)	75.092 (13.180)
hex_value(#1)	46354	3.157	9.233 (2.092)	4.412 (0.370)	2.333 (0.128)	4.821 (1.722)	3.894 (1.471)
transform_from_basic_ops(#10)	40169	10.253	115.732 (8.667)	9.020 (0.452)	1.552 (0.079)	106.712 (8.215)	75.875 (5.514)
get_descriptor_length_24b(#1)	46625	5.442	11.687 (0.718)	7.791 (0.329)	2.384 (0.104)	3.896 (0.388)	1.988 (0.301)
tile_pos(#4)	24696	8.031	67.636 (27.126)	7.045 (0.397)	1.756 (0.091)	60.591 (26.728)	46.309 (20.400)
diract_picture_n_before_m(#2)	46393	6.615	15.315 (1.327)	6.968 (0.315)	2.226 (0.116)	8.347 (1.012)	3.746 (0.337)
ps_id_to_tk(#1)	46721	4.41	15.811 (2.370)	7.414 (1.090)	2.579 (0.190)	8.397 (1.280)	6.504 (1.127)
leading_zero_count(#1)	46727	7.838	16.737 (2.105)	8.462 (0.598)	2.090 (0.136)	8.275 (1.507)	3.473 (0.609)
trailing_zero_count(#1)	46701	3.392	19.508 (6.189)	4.161 (0.706)	1.881 (0.135)	15.347 (5.483)	13.786 (5.088)
popcnt_32(#1)	46802	5.602	11.500 (0.818)	7.296 (0.313)	2.471 (0.155)	4.204 (0.504)	2.076 (0.335)
parity(#1)	46821	4.988	9.968 (0.644)	6.447 (0.292)	2.584 (0.179)	3.521 (0.352)	1.813 (0.244)
dv_audio_12_to_16(#1)	46637	3.884	17.708 (2.780)	8.279 (0.598)	3.607 (0.155)	9.429 (2.182)	7.004 (1.673)
is_power_2(#1)	46801	3.791	9.130 (1.357)	5.420 (0.316)	2.819 (0.225)	3.710 (1.042)	2.218 (0.659)
RenderRGB(#3)	46061	5.663	17.038 (0.901)	9.718 (0.366)	2.670 (0.172)	7.320 (0.535)	4.023 (0.330)
decode_BCD(#1)	46824	4.706	8.751 (1.124)	5.516 (0.356)	1.890 (0.202)	3.235 (0.768)	1.903 (0.618)
mpga_get_frame_samples(#1)	46235	3.366	9.288 (2.057)	4.887 (0.497)	2.580 (0.148)	4.401 (1.560)	3.595 (1.454)

TABLE V: Metrics for the timeout conclusion for all reference functions

function name	#	steps	total time (solver)	CE total time (solver)	CE last time (solver)	AS total time (solver)	AS last time (solver)	AS-stops/ CE-stops
clamp	5595	16.505	300.000 (44.278)	27.856 (8.112)	9.392 (6.966)	272.144 (36.167)	140.702 (17.457)	5416/179
prev_pow_2(#1)	32	1	300.000 (289.445)	300.000 (289.445)	300.000 (289.445)	0.000 (0.000)	0.000 (0.000)	0/32
abs_diff(#2)	6	5.667	300.000 (286.525)	297.333 (286.318)	288.167 (285.378)	2.667 (0.206)	1.167 (0.112)	0/6
bswap32(#1)	8	2.75	300.000 (293.526)	299.125 (293.479)	296.250 (293.329)	0.875 (0.047)	0.875 (0.047)	0/8
integer_cmp(#2)	271	3.085	300.000 (247.247)	296.347 (246.627)	288.122 (243.312)	3.653 (0.620)	1.063 (0.209)	3/268
even(#1)	5	1.8	300.000 (116.452)	299.600 (116.434)	297.400 (116.320)	0.400 (0.019)	0.400 (0.019)	0/5
div255(#1)	4	2.5	300.000 (294.241)	299.500 (294.203)	297.500 (294.115)	0.500 (0.037)	0.500 (0.037)	0/4
reverse_bits(#1)	14	3	300.000 (292.294)	298.714 (292.182)	294.786 (291.965)	1.286 (0.112)	1.286 (0.112)	0/14
binary_log(#1)	255	1.239	300.000 (207.291)	298.824 (206.920)	277.769 (203.879)	1.176 (0.371)	0.949 (0.336)	19/236
median(#3)	14328	13.634	300.000 (65.444)	15.655 (2.144)	3.266 (1.319)	284.345 (63.300)	167.910 (35.663)	14184/144
hex_value(#1)	477	1.013	300.000 (268.765)	299.964 (268.754)	298.753 (268.165)	0.036 (0.010)	0.027 (0.007)	2/475
transform_from_ basic_ops(#10)	6409	18.381	300.000 (27.949)	22.098 (3.092)	4.510 (2.408)	277.902 (24.857)	172.895 (14.278)	6319/90
get_descriptor_ length_24b(#1)	184	1.391	300.000 (233.380)	299.832 (233.373)	298.853 (233.277)	0.168 (0.006)	0.168 (0.006)	0/184
tile_pos(#4)	16518	7.634	300.000 (280.532)	8.118 (1.326)	2.782 (0.988)	291.882 (279.206)	256.574 (249.372)	16441/77
dirac_picture_ n_before_m(#2)	108	51.481	300.000 (137.988)	132.556 (87.679)	89.917 (85.144)	167.444 (50.309)	25.954 (3.204)	74/34
ps_id_to_tk(#1)	110	1.118	300.000 (258.764)	299.755 (258.748)	291.764 (250.903)	0.245 (0.015)	0.218 (0.014)	3/107
leading_zero_ count(#1)	63	5.079	300.000 (143.608)	297.254 (143.230)	171.063 (111.259)	2.746 (0.379)	0.841 (0.100)	1/62
trailing_zero_ count(#1)	84	1.476	300.000 (283.679)	299.155 (283.545)	285.417 (270.053)	0.845 (0.134)	0.643 (0.111)	4/80
popcnt_32(#1)	29	1	300.000 (295.366)	300.000 (295.366)	300.000 (295.366)	0.000 (0.000)	0.000 (0.000)	0/29
parity(#1)	10	1.2	300.000 (266.296)	299.900 (266.293)	275.100 (266.280)	0.100 (0.003)	0.100 (0.003)	0/10
dv_audio_ 12_to_16(#1)	194	1.026	300.000 (290.336)	299.979 (290.334)	296.928 (288.827)	0.021 (0.002)	0.021 (0.002)	1/193
is_power_2(#1)	30	3.667	300.000 (291.082)	297.833 (290.309)	293.867 (290.012)	2.167 (0.773)	1.133 (0.375)	0/30
RenderRGB(#3)	7	4.429	300.000 (115.721)	297.000 (115.538)	290.714 (115.275)	3.000 (0.184)	1.714 (0.099)	0/7
decode_BCD(#1)	7	1.857	300.000 (126.084)	299.714 (126.040)	298.429 (125.986)	0.286 (0.044)	0.286 (0.044)	0/7
mpga_get_ frame_samples(#1)	574	1.024	300.000 (289.464)	299.963 (289.460)	297.423 (288.201)	0.037 (0.003)	0.035 (0.003)	4/570