SynGuar: Guaranteeing Generalization in Programming by Example (Appendix)

A PROOFS

In this section, we give the full proofs for the sample complexity bound and correctness of the StrSTUN counting phase.

Claim A.1 (P3). SynGuar *uses no more than* $2\omega(Q)$ *examples on any* Q *when the result is not None with* $g(x) = g_0(x) = \max\{0,\ \frac{1}{\epsilon}(\ln(x) - \ln(\frac{1}{\delta}))\}$ *and* $k \leq \frac{1}{2\epsilon}\ln\frac{1}{\delta}$.

Proof. Let $S' \subset Q$ be the samples in sampling phase. Let P be the samples when $\omega(Q) = \textsc{Prefix}(Q, g)$ for some g and let us call this the best stopping point for SynGuar's sampling phase on Q. Then $\omega(Q) = |P| + \frac{1}{\epsilon}(\ln|H_P| + ln\frac{1}{\delta})$. Let $g_0(x) = \max\{0,\ \frac{1}{\epsilon}(\ln(x) - \ln(1/\delta))\}$, $\gamma(Q) = \textsc{Prefix}(Q, g_0)$ and $S' \subset Q$ be the samples in sampling phase.

If P is the samples for the best stopping point, then

$$\omega(Q) = |P| + \frac{1}{\epsilon}(\ln|H_P| + ln\frac{1}{\delta})$$

Case 1: SynGuar using g_0 is stopping earlier in phrase 1 than the best possible stopping point, $|S'| \leq |P|$

$$\begin{aligned}\gamma(Q) - 2\cdot\omega(Q) &= |S'| - 2\cdot|P| + \frac{1}{\epsilon}\cdot(\ln|H_{S'}| - 2\cdot\ln|H_P|) \\ &\quad - \frac{1}{\epsilon}\cdot\ln\frac{1}{\delta} \\ &\leq -|P| - \frac{1}{\epsilon}\cdot\ln\frac{1}{\delta} + \frac{1}{\epsilon}\cdot(\ln|H_{S'}|) \\ &\quad \text{since } |S'| \leq |P|\end{aligned}$$

Observe that, for $g_0(x) = \max\{0,\ \frac{1}{\epsilon}(\ln(x) - \ln(1/\delta))\}$ the $|S'| \geq \frac{1}{\epsilon}\cdot(\ln|H_{S'}| - \ln\frac{1}{\delta})$ (see, step 12 in Algorithm 2)

$$\gamma(Q) - 2\cdot\omega(Q) \leq -|P| + |S'| \leq 0$$

Case 2: SynGuar using g_0 is stopping after the best possible stopping point in phrase 1, $|S'| > |P|$

In this case, $|H_P| \geq |H_{S'}|$. Observe that $|S'| \leq \omega(Q)$. Because for SynGuar using g_0, after seeing P, it will take no more than $g_0(|H_P|) + 2k = \max\{0, \frac{1}{\epsilon}(\ln|H_P| - ln\frac{1}{\delta})\} + 2k \leq \frac{1}{\epsilon}(\ln|H_P| + ln\frac{1}{\delta})$ examples in phase 1.

$$\begin{aligned}|S'| &\leq |P| + 2k + g_0(|H_P|) \\ \omega(Q) &= |P| + \frac{1}{\epsilon}(\ln|H_P| + ln\frac{1}{\delta}) \\ &\implies |S'| \leq \omega(Q)\end{aligned}$$

Now,

$$\begin{aligned}\gamma(Q) - 2\cdot\omega(Q) &= |S'| + \frac{1}{\epsilon}\cdot(\ln|H_{S'}| + \ln\frac{1}{\delta}) - 2\cdot\omega(Q) \\ &\leq \frac{1}{\epsilon}\cdot(\ln|H_{S'}| + \ln\frac{1}{\delta}) - \omega(Q) \quad (\text{by } |S'| \leq \omega(Q)) \\ &\leq -|P| + \frac{1}{\epsilon}\cdot(\ln|H_{S'}| - \ln|H_P|) \\ &\leq 0 \quad (\text{since, } |H_P| \geq |H_{S'}|)\end{aligned}$$

□

Claim A.2. StrSTUN *computes a sound upper bound of hypothesis size that is consistent with the seen inputs* (H_S).

Proof. We prove that the counting rules for value vector $\mathbf{v}$, consistency vector $\mathbf{c}$, and also succinct representation of consistency vectors C in Table 1 of the paper produce sound upper bounds of the corresponding program sets. Then the claim above is true.

Claim A.2.1: When $\text{Count}(\mathbf{v}, t)$ is updated, $\forall v', \forall t' < t, \text{Count}(\mathbf{v}', t')$ will no longer be updated.

The enumeration is from the smallest component size to the maximum component size. If some $\text{Count}(\mathbf{v}', t')$ is updated after $\text{Count}(\mathbf{v}, t)$ is updated, means the algorithm enumerates program with component size $t' < t$ after enumerate a program with component size t. According to the property of the enumeration phase, this is not possible.

Claim A.2.2: After finishing enumerating all programs at component size t, $\forall \mathbf{v}$, $\text{Count}(\mathbf{v}, t)$ is a sound upper bound of number of programs that in component size t and have value vector $\mathbf{v}$.

By induction. The $\text{Count}(\mathbf{v_{base}}, 0)$ for the syntactic terminals is a direct enumeration which is sound. According to claim A.2.1, $\forall t' < t, \forall \mathbf{v}', \text{Count}(\mathbf{v}', t')$ will no longer be updated after enumerating component size $t-1$. Assume they are sound by induction, then after the enumeration of component size t, $\text{Count}(\mathbf{v}, t)$ is at least sum of all syntactically different programs that produce $\mathbf{v}$ according to the update rule (2) in Table 1, which is also a sound upper bound of number of programs at component size t have value vector $\mathbf{v}$.

Claim A.2.3: After the enumeration, $\text{Count}(\mathbf{c})$ is a sound upper bound of the number of programs under component size limit that have consistency vector $\mathbf{c}$.

During the enumeration phase, all the programs under component size limit is clustered by value vector $\mathbf{v}$ by component size t with a sound upper bound for $\mathbf{v}$ at component size t counted as $\text{Count}(\mathbf{v}, t)$. Clustering them to consistency vectors and sum up according to rule (3) is still a sound upper bound.

ESEC/FSE 2021, 23 - 27 August, 2021, Athens, Greece
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Target programs in benchmark for SynGuar-PROSE

index	description	example
P1	lowercase of the first word	"HELLO WORLD" → "hello"
P2	second word	"HELLO WORLD" → "WORLD"
P3	abbreviation of 2 words	"hello word" → "H. W"
P4	2nd last seperated text	"a3-ge-7r-32" → "7r"
P5	add parentheses	"sin x" → "sin(x)"
P6	abbreviation of 3 words	"Unreal Engine 4" → "ue4"
P7	change name format	"Second First" → "FIRST Second"
P8	software name formatting	"Company App 5" → "App(5) by COMPANY"
P9	change date format	"2020-11-12" → "12/11/2020"
P10	reverse first 8 characters	"12345678" → "87654321"
P11	bolden the first capital word	"please LEAVE!" → "please LEAVE!"
P12	trim and capitalize first letter	" big-bang theory " → "Big-bang theory"
P13	extract number	"weight is 56.7kg" → "56.7"
P14	capitalize and prepend	"phone" → "iPhone"
P15	bash command shorthand	" ssh tom" → "ssh tom@univ.edu"
P16	extract last 4 capital letters	"go As Soon As Possible" → "ASAP"

Claim A.2.4: In the unification phase, the number of programs that contain and only contains k conditions and whose output behaviour is consistent with a succinct representation of consistency vectors C is soundly upper-bounded by $\texttt{Count}(C, k)$.

By induction. According to rule (4), the number of programs that have 0 conditions (straight-line program) is just the sum of counts on the set of consistency vectors is soundly upper bounded by $\texttt{Count}(C, 0)$ because of Claim A.2.3.

According to rule (5) in table 1, $\texttt{Count}(C, k)$ will be reduced to the sum of the count for each cluster of conditions that have the same value vector b, all syntactically different combinations of program in `then` and `else` branch, while the number of conditions in `then` branch plus `else` branch plus 1 is k. In other words, $\texttt{Count}(C, k)$ is reduced to subproblems in the form of $\texttt{Count}(C', k')$, where $k' < k$. In other words, Rule (5) will always reduce to counting with smaller number of conditions on some C' for `else` and `then` branch. By induction it is a sound upper bound.

□

B PROSE-BENCHMARK PROGRAMS

We describe the functionality of the programs in the PROSE-benchmark in Table 1. We give the target program implementation of the 16 benchmark programs in the PROSE-benchmark in Figure 1.

C SYGUS-STUN PROGRAMS

As stated in our evaluation, our SyGuS-STUN consists of 59 programs from the SyGuS 2019 PBE-string track benchmark. For brevity, in the paper we refer to these by an index instead of the names of the programs. The names of the 59 benchmark programs for SynGuar-STUN is listed in Table 3.

D PROSE IMPLEMENTATION DETAILS

In Figure 2, we show the executable semantics of the operators that we added in our DSL or slightly modified from the default available DSL in PROSE. On top of the common operators (`Concat`, `SubStr`, `AbsPos` and `ConstStr`) we implement witness functions for a slightly different relative position operator `RegPos` and add two more operators, `LowerCase` and `UpperCase`, respectively, to the FlashFill DSL in the PROSE framework. They closely resemble the inverse semantics for the DSL of FlashFill [2]. In Figure 3, we show the witness functions for the `UpperCase` and `LowerCase` operators. For the implementation details for all the operators, please see the source code of the artifact [1].

```
1. LowerCase(SubString(x, RelPos(x, (^\s*, .*?, 0, 0)), RelPos(x, (\w+, .*?, 0, 0))), x)
2. SubString(x, RelPos(x, (.*?, \w+, 1, 0)), RelPos(x, (\w+, .*?, 1, 0)))
3. Concats(SubString(x, RelPos(x, (.*?, [A-Z]+, 0, 0)), RelPos(x, (.*?, [A-Z]+, 0, 1))),
       Concats(ConstStr(". "), SubString(x, RelPos(x, (.*?, [A-Z]+, 1, 0)), RelPos(x,
       (.*?, [A-Z]+, 1, 1)))))
4. SubString(x, RelPos(x, (.*?, [\,\.\;\-\|], -2, 1)), RelPos(x, (.*?, [\,\.\;\-\|], -1,
       0)))
5. Concats(SubString(x, RelPos(x, (^\s*, .*?, 0, 0)), RelPos(x, (\w+, .*?, 0, 0))),
       Concats(ConstStr("("), Concats(SubString(x, RelPos(x, (\w+, .*?, 0, 1)), RelPos(x
       , (.*?, \s*$, 0, 0))), ConstStr(")"))))
6. Concats(LowerCase(SubString(x, RelPos(x, (.*?, \w+, 0, 0)), RelPos(x, (.*?, \w+, 0, 1)
       )), x), Concats(LowerCase(SubString(x, RelPos(x, (.*?, \w+, 1, 0)), RelPos(x,
       (.*?, \w+, 1, 1))), x), SubString(x, RelPos(x, (.*?, \w+, -1, 0)), RelPos(x,
       (.*?, \s*$, 0, 0)))))
7. Concats(UpperCase(SubString(x, RelPos(x, (.*?, \w+, 1, 0)), RelPos(x, (\w+, .*?, 1, 0)
       )), x), Concats(ConstStr(" "), SubString(x, RelPos(x, (.*?, \w+, 0, 0)), RelPos(x
       , (\w+, .*?, 0, 0)))))
8. Concats(SubString(x, RelPos(x, (.*?, \w+, 1, 0)), RelPos(x, (\w+, .*?, 1, 0))),
       Concats(ConstStr("("), Concats(SubString(x, RelPos(x, (.*?, \d+, 0, 0)), RelPos(x
       , (\d+, .*?, 0, 0))), Concats(ConstStr(") by "), UpperCase(SubString(x, RelPos(x,
        (.*?, \w+, 0, 0)), RelPos(x, (\w+, .*?, 0, 0))), x)))))
9. Concats(SubString(x, RelPos(x, (.*?, \d+, 2, 0)), RelPos(x, (\d+, .*?, 2, 0))),
       Concats(ConstStr("/"), Concats(SubString(x, RelPos(x, (.*?, \d+, 1, 0)), RelPos(x
       , (\d+, .*?, 1, 0))), Concats(ConstStr("/"), SubString(x, RelPos(x, (.*?, \d+, 0,
        0)), RelPos(x, (\d+, .*?, 0, 0)))))))
10. Concats(SubString(x, AbsPos(x, 7), AbsPos(x, 8)), Concats(SubString(x, AbsPos(x, 6),
       AbsPos(x, 7)), Concats(SubString(x, AbsPos(x, 5), AbsPos(x, 6)), Concats(
       SubString(x, AbsPos(x, 4), AbsPos(x, 5)), Concats(SubString(x, AbsPos(x, 3),
       AbsPos(x, 4)), Concats(SubString(x, AbsPos(x, 2), AbsPos(x, 3)), Concats(
       SubString(x, AbsPos(x, 1), AbsPos(x, 2)), SubString(x, AbsPos(x, 0), AbsPos(x, 1)
       ))))))))
11. Concats(SubString(x, AbsPos(x, 0), RelPos(x, (.*?, [A-Z]+, 0, 0))), Concats(ConstStr(
       "<b>"), Concats(SubString(x, RelPos(x, (.*?, [A-Z]+, 0, 0)), RelPos(x, ([A-Z]+,
       .*?, 0, 0))), Concats(ConstStr("</b>"), SubString(x, RelPos(x, ([A-Z]+, .*?, 0,
       0)), AbsPos(x, -1))))))
12. Concats(UpperCase(SubString(x, RelPos(x, (^\s*, .*?, 0, 0)), RelPos(x, (^\s*, .*?, 0,
        1))), x), LowerCase(SubString(x, RelPos(x, (^\s*, .*?, 0, 1)), RelPos(x, (.*?, \
       s*$, 0, 0))), x))
13. SubString(x, RelPos(x, (.*?, (-?\d+)(\.\d+)?, 0, 0)), RelPos(x, ((-?\d+)(\.\d+)?,
       .*?, 0, 0)))
14. Concats(ConstStr("i"), Concats(UpperCase(SubString(x, RelPos(x, (.*?, \w+, 0, 0)),
       RelPos(x, (.*?, \w+, 0, 1))), x), SubString(x, RelPos(x, (.*?, \w+, 0, 1)),
       RelPos(x, (\w+, .*?, 0, 0)))))
15. Concats(SubString(x, RelPos(x, (.*?, \w+, 0, 0)), RelPos(x, (\w+, .*?, 0, 1))),
       Concats(SubString(x, RelPos(x, (.*?, \w+, -1, 0)), RelPos(x, (\w+, .*?, -1, 0))),
        Concats(ConstStr("@univ.edu"), SubString(x, RelPos(x, (\w+, .*?, 0, 0)), RelPos(
       x, (.*?, \w+, -1, 0))))))
16. Concats(SubString(x, RelPos(x, (.*?, [A-Z]+, -4, 0)), RelPos(x, (.*?, [A-Z]+, -4, 1))
       ), Concats(SubString(x, RelPos(x, (.*?, [A-Z]+, -3, 0)), RelPos(x, (.*?, [A-Z]+,
       -3, 1))), Concats(SubString(x, RelPos(x, (.*?, [A-Z]+, -2, 0)), RelPos(x, (.*?, [
       A-Z]+, -2, 1))), SubString(x, RelPos(x, (.*?, [A-Z]+, -1, 0)), RelPos(x, (.*?, [A
       -Z]+, -1, 1))))))
```

Figure 1: String-manipulation programs in the PROSE-benchmark **used to evaluate** SynGuar-PROSE**.**

E REDUCTION IN THE HYPOTHESIS SPACE REDUCTION

We also measure the consistent program space which remains when SynGuar-PROSE terminates. It varies for different benchmarks between 10^0-10^6 (not shown). This is because some programs behave quite similar or even exactly the same on our I/O distribution (for example, there is no "\n" character in our inputs, which makes `AbsPos(x, 0)` and `RelPos(x, (^\s*, .*?, 0, 0))` equivalent). The detail of the reduction in hypothesis space on the first 30 examples averaged over 32 runs (logarithmic mean) is in 2.

```
string ConstStr(string x) { return x; }
string UpperCase(string term, string x) {
  if (x.IndexOf(term) >= 0) return term.ToUpper();
  else return null; }

string LowerCase(string term, string x) {
  if (x.IndexOf(term) >= 0) return term.ToLower();
  else return null; }

int? AbsPos(string x, int ka) {
  if (ka > x.Length || ka < -x.Length - 1) return null;
  return ka >= 0 ? ka : x.Length + ka + 1; }

int? RegPos(string x, Tuple<Regex, Regex, int, int> kr) {
  MatchCollection leftMatches = kr.Item1.Matches(x);
  MatchCollection rightMatches = kr.Item2.Matches(x);
  int k = kr.Item3;
  int offset = kr.Item4;

  for (leftMatch in leftMatches)
    for (rightMatch in rightMatches)
      if (rightMatch.Index == leftMatch.Index + leftMatch.Length)
        matchPos.Add(rightMatch.Index);

  if (k >= matchPos.Count || k < -matchPos.count)
    return null;
  int selPos = k >= 0 ? matchPos[k] : matchPos[matchPos.Count + k];
  int finalPos = selPos + offset;
  if (finalPos > x.Length || finalPos < 0) return null;
  return finalPos; }
```

Figure 2: Executable semantics of the operators of SynGuar, defined by the DSL. We only show the modified/newly added functions.

$$\text{LowerCase}(term, x) : \omega_{term}(\sigma \rightsquigarrow w | x = v)$$
$$= \sigma \rightsquigarrow \bigvee_{w \text{ occurs in lowercase of } v \text{ at position } l} \langle l, l + |w| \rangle$$
$$\text{UpperCase}(term, x) : \omega_{term}(\sigma \rightsquigarrow w | x = v)$$
$$= \sigma \rightsquigarrow \bigvee_{w \text{ occurs in uppercase of } v \text{ at position } l} \langle l, l + |w| \rangle$$

Figure 3: Witness functions ω_{term} of the operators of SynGuar, defined by the DSL. (σ, s) represents the I/O examples (the inductive specification).

REFERENCES

[1] 2021. *SynGuar Artifact.* https://github.com/HALOCORE/SynGuar
[2] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, Vol. 50. 107–126. https://doi.org/10.1145/2858965.2814310

Table 2: The reduction of hypothesis space on SynGuar-PROSE (logarithmic mean over 32 runs) for the first 30 examples

sample	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
1	5.8E+15	2.2E+11	1.8E+10	4.1E+12	1.9E+37	1.2E+20	3.0E+22	1.7E+36	2.8E+13	5.3E+28	6.5E+43	2.5E+39	9.7E+03	4.9E+15	2.5E+50	2.6E+14
2	7.9E+04	8.5E+03	1.6E+05	1.6E+04	3.3E+15	1.9E+07	4.9E+09	5.5E+15	7.1E+07	6.7E+12	8.3E+16	4.1E+20	1.6E+02	1.4E+07	2.1E+19	1.2E+07
3	2.8E+02	1.2E+02	1.3E+03	4.3E+02	2.4E+09	8.9E+03	6.4E+04	1.0E+10	9.9E+05	9.6E+07	1.1E+10	3.2E+14	5.2E+01	1.3E+04	2.7E+11	1.3E+04
4	5.6E+01	7.4E+00	1.7E+02	7.5E+01	3.5E+07	5.7E+02	2.9E+03	3.3E+07	3.1E+05	8.6E+05	4.1E+07	8.7E+11	3.8E+01	1.4E+03	4.5E+07	7.4E+02
5	1.4E+01	4.0E+00	4.6E+01	2.2E+01	3.4E+06	2.1E+02	6.6E+02	1.3E+06	1.5E+05	1.5E+05	6.4E+06	2.8E+10	2.7E+01	2.8E+02	2.2E+06	2.6E+02
6	1.0E+01	2.3E+00	3.2E+01	1.1E+01	3.3E+05	1.0E+02	1.3E+02	2.3E+05	5.7E+04	5.1E+04	3.6E+06	5.9E+09	2.5E+01	1.1E+02	4.9E+05	1.6E+02
7	7.8E+00	1.8E+00	2.5E+01	1.1E+01	4.7E+04	4.5E+01	5.3E+01	7.1E+04	4.2E+04	2.1E+04	1.9E+06	1.2E+09	2.3E+01	5.3E+01	1.6E+05	1.1E+02
8	7.1E+00	1.4E+00	2.4E+01	8.6E+00	1.6E+04	3.1E+01	3.3E+01	1.8E+04	3.0E+04	1.8E+04	7.5E+05	3.8E+08	2.0E+01	3.3E+01	7.7E+04	6.7E+01
9	6.5E+00	1.2E+00	2.2E+01	8.0E+00	2.4E+03	1.3E+01	2.2E+01	9.6E+03	1.7E+04	7.6E+03	5.0E+05	1.5E+08	1.9E+01	1.9E+01	3.6E+04	5.6E+01
10	4.9E+00	1.2E+00	1.9E+01	5.3E+00	1.1E+03	9.8E+00	1.3E+01	5.8E+03	1.3E+04	7.0E+03	3.4E+05	7.5E+07	1.9E+01	1.4E+01	2.3E+04	4.1E+01
11	3.7E+00	1.2E+00	1.9E+01	5.2E+00	5.4E+02	8.6E+00	1.2E+01	5.0E+03	8.7E+03	5.1E+03	1.8E+05	3.0E+07	1.7E+01	8.0E+00	1.7E+04	3.3E+01
12	3.5E+00	1.1E+00	1.6E+01	5.2E+00	4.2E+02	6.6E+00	8.4E+00	4.7E+03	6.2E+03	4.1E+03	1.3E+05	1.6E+07	1.6E+01	7.9E+00	1.2E+04	2.5E+01
13	3.4E+00	1.0E+00	1.5E+01	4.9E+00	2.7E+02	6.6E+00	8.1E+00	3.3E+03	4.2E+03	3.9E+03	1.3E+05	1.5E+07	1.4E+01	6.8E+00	9.7E+03	2.4E+01
14	3.0E+00	1.0E+00	1.4E+01	4.7E+00	1.4E+02	4.7E+00	7.6E+00	2.3E+03	3.7E+03	2.9E+03	1.1E+05	8.2E+06	1.4E+01	4.3E+00	8.7E+03	2.0E+01
15	2.8E+00	1.0E+00	1.1E+01	4.7E+00	1.2E+02	4.0E+00	6.8E+00	2.1E+03	2.7E+03	2.4E+03	1.1E+05	5.2E+06	1.3E+01	3.7E+00	8.1E+03	1.9E+01
16	2.6E+00	1.0E+00	1.1E+01	4.5E+00	8.3E+01	4.0E+00	6.2E+00	1.3E+03	2.5E+03	2.4E+03	6.8E+04	4.3E+06	1.3E+01	3.2E+00	8.1E+03	1.7E+01
17	2.6E+00	1.0E+00	1.0E+01	4.5E+00	4.8E+01	4.0E+00	5.6E+00	1.1E+03	1.9E+03	2.3E+03	4.0E+04	2.0E+06	1.3E+01	3.2E+00	7.1E+03	1.7E+01
18	2.5E+00	1.0E+00	9.6E+00	4.5E+00	3.3E+01	3.6E+00	5.3E+00	8.9E+02	1.6E+03	2.3E+03	3.9E+04	1.2E+06	1.2E+01	3.2E+00	7.1E+03	1.7E+01
19	2.5E+00	1.0E+00	9.1E+00	4.2E+00	2.5E+01	3.6E+00	4.7E+00	6.7E+02	1.5E+03	2.3E+03	3.0E+04	1.1E+06	1.2E+01	3.0E+00	6.6E+03	1.7E+01
20	2.5E+00	1.0E+00	9.1E+00	4.2E+00	1.8E+01	3.3E+00	4.6E+00	5.8E+02	1.2E+03	2.2E+03	2.6E+04	6.2E+05	1.2E+01	2.6E+00	4.7E+03	1.7E+01
21	2.5E+00	1.0E+00	9.1E+00	4.2E+00	1.3E+01	3.2E+00	3.4E+00	4.7E+02	1.1E+03	2.2E+03	2.3E+04	4.9E+05	1.2E+01	2.6E+00	4.0E+03	1.7E+01
22	2.5E+00	1.0E+00	8.7E+00	4.2E+00	1.3E+01	3.1E+00	3.2E+00	4.0E+02	1.0E+03	2.2E+03	2.1E+04	4.8E+05	1.1E+01	2.2E+00	3.8E+03	1.7E+01
23	2.5E+00	1.0E+00	8.7E+00	4.2E+00	1.2E+01	2.6E+00	2.9E+00	4.0E+02	9.8E+02	2.2E+03	1.7E+04	3.8E+05	1.1E+01	1.6E+00	3.8E+03	1.7E+01
24	2.5E+00	1.0E+00	8.7E+00	4.2E+00	1.1E+01	2.3E+00	2.8E+00	3.8E+02	9.4E+02	2.2E+03	1.7E+04	3.2E+05	1.1E+01	1.6E+00	3.8E+03	1.7E+01
25	2.5E+00	1.0E+00	8.6E+00	4.2E+00	1.0E+01	2.2E+00	2.8E+00	3.8E+02	9.2E+02	2.2E+03	1.7E+04	3.1E+05	1.1E+01	1.6E+00	3.4E+03	1.6E+01
26	2.3E+00	1.0E+00	8.6E+00	4.2E+00	9.7E+00	1.9E+00	2.8E+00	3.8E+02	8.7E+02	2.2E+03	1.6E+04	1.5E+05	1.1E+01	1.4E+00	2.9E+03	1.6E+01
27	2.3E+00	1.0E+00	8.6E+00	4.2E+00	7.9E+00	1.9E+00	2.6E+00	3.5E+02	7.3E+02	2.2E+03	1.6E+04	1.4E+05	1.1E+01	1.4E+00	2.6E+03	1.6E+01
28	2.3E+00	1.0E+00	8.6E+00	4.0E+00	6.2E+00	1.9E+00	2.6E+00	3.3E+02	6.6E+02	2.2E+03	1.6E+04	8.6E+04	1.1E+01	1.4E+00	2.5E+03	1.6E+01
29	2.2E+00	1.0E+00	8.6E+00	4.0E+00	5.7E+00	1.9E+00	2.4E+00	3.1E+02	5.9E+02	2.2E+03	1.6E+04	8.5E+04	1.1E+01	1.4E+00	2.5E+03	1.6E+01
30	2.2E+00	1.0E+00	8.6E+00	4.0E+00	5.1E+00	1.7E+00	2.4E+00	2.5E+02	5.9E+02	2.2E+03	1.6E+04	7.4E+04	1.1E+01	1.4E+00	2.4E+03	1.6E+01

Table 3: Benchmarks for SynGuar-STUN: program id and corresponding names in SyGuS 2019 (euphony) benchmark.

index	benchmark_name
S1	19558979
S2	28627624_1
S3	30732554
S4	37534494
S5	38664547
S6	38871714
S7	add-a-line-break-with-a-formula
S8	change-negative-numbers-to-positive
S9	convert-numbers-to-text
S10	convert-text-to-numbers
S11	count-specific-characters-in-a-cell
S12	count-total-characters-in-a-cell
S13	create-email-address-from-name
S14	create-email-address-with-name-and-domain
S15	exceljet1
S16	exceljet2
S17	exceljet4
S18	find-nth-occurrence-of-character
S19	get-domain-from-email-address_2
S20	get-first-name-from-name
S21	get-last-name-from-name
S22	get-last-word
S23	join-first-and-last-name
S24	most-frequently-occurring-text
S25	phone-5-short
S26	phone-6-short
S27	phone-7-short
S28	position-of-2nd-3rd-etc-instance-of-character
S29	remove-characters-from-left
S30	remove-characters-from-right

index	benchmark_name
S31	remove-file-extension-from-filename
S32	remove-text-by-matching
S33	remove-text-by-position
S34	remove-unwanted-characters
S35	replace-one-character-with-another
S36	split-numbers-from-units-of-measure_1
S37	split-numbers-from-units-of-measure_2
S38	split-text-and-numbers
S39	stackoverflow1
S40	stackoverflow10
S41	stackoverflow2
S42	stackoverflow3
S43	stackoverflow8
S44	stackoverflow9
S45	strip-html-from-text-or-numbers
S46	33619752
S47	35016216
S48	35744094
S49	41503046
S50	44789427
S51	count-line-breaks-in-cell
S52	get-domain-name-from-url
S53	get-first-name-from-name-with-comma
S54	get-first-word
S55	remove-leading-and-trailing-spaces-from-text
S56	split-text-string-at-specific-character
S57	stackoverflow7
S58	strip-non-numeric-characters
S59	strip-numeric-characters-from-cell