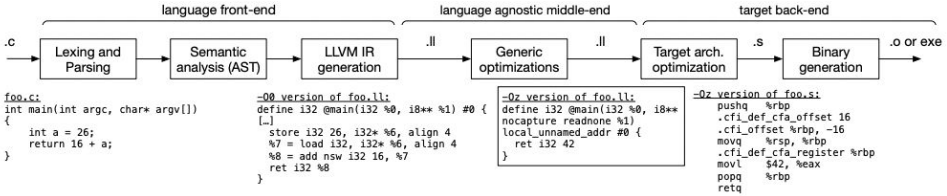
Code Translation with Compiler Representations

Marc Szafraniec
Baptiste Rozière
Hugh Leather
François Charton
Patrick Labatut
Gabriel Synnaeve
Meta AI

ML4Code Reading Group - October 27th, 2022 - Avinash

Introduction

- Neural Machine Translation (NMT) is used to solve the problem of code translation
  - Reliability issues
  - Unavailability of parallel training data
  - TransCoder: Unsupervised code translation
- Compilers like LLVM create Intermediary Representations (IR) which are language agnostic



Data

- GitHub repositories using Google BigQuery

	C++	Go	Java	Rust
Monolingual data	6.6 M	9.4 M	7.8 M	576.3 K
Code / IR Parallel Data	344.4 K	384.4 K	2.2 M	19.2 K
Successful IR Compilation	5.2%	4.1%	28.2%	3.3%

- IRs are generated uniformly across the languages (-Oz flag), strip header and footer, debug information and comments, block names are canonicalized and symbol names are demangled.
- Evaluated using CA@1 metric on computational accuracy test suite (same one as TransCoder)

Training

- Masked Language Modeling (MLM)
- Denoising Auto Encoding (AE)
- Back Translation (BT)
- Translation Language Modeling (TLM)
- Translation Auto Encoding (TAE)
- IR Generation (MT)
- IR decompilation
- IR pivot translation

Results

	from C++	to C++	from Go	to Go	from Java	to Java	from Rust	to Rust	AVG
Greedy decoding									
IR Pivot	17.4	24.0	19.9	11.5	11.9	22.2	16.3	7.8	16.4
TransCoder (baseline)	46.4	52.1	42.1	45.6	41.2	44.5	29.6	17.0	39.8
TLM	47.5	54.8	45.4	41.2	39.8	**52.1**	31.1	15.7	40.9
MLM + TAE	47.3	53.3	**47.2**	44.8	41.8	45.9	25.1	17.4	40.4
TLM + TAE	46.9	**55.9**	45.0	37.9	38.5	54.5	34.9	16.8	41.3
MLM + MT	45.5	51.0	44.0	48.9	46.6	45.2	25.7	16.6	40.5
TLM + MT	45.6	51.5	45.1	47.1	46.9	45.5	24.4	17.9	40.5
TLM + TAE + MT	**47.8**	54.3	46.6	**51.6**	**47.1**	49.6	**35.3**	**21.4**	**44.2**
Beam size 5									
TransCoder (baseline)	**53.8**	53.4	45.2	54.4	46.1	51.5	35.9	20.9	45.3
TLM + TAE + MT	52.9	**53.5**	**48.8**	**57.1**	**51.5**	**53.4**	**37.9**	**27.1**	**47.8**

Discussion

- Representation of interpreted languages
- Pivot works well for low resource languages because the IR can be computed using a ruled based compiler
- IR Computation is not required at inference time

Examples

Input function	TransCoder	TransCoder-IR

`// Java`
`static int addOne ( int x ) {`
`  return ( - ( ~ x ) ) ;`
`}`
	`// Rust`
	`fn add_one (x : int ) -> int {`
	`  (- ( ~ x) as int)`
	`}`
		`// Rust`
		`fn add_one (x : i32 ) -> i32 {`
		`  (- ( ! x))`
		`}`

`// Java`
`static boolean isEven (int n){`
`  return ( n % 2 == 0 );`
`}`
	`// Rust`
	`fn is_even ( n : u32 ) -> bool`
	`  ( n % 2 == 0 ) as bool`
	`}`
		`// Rust`
		`fn is_even ( n : i32 ) -> bool {`
		`  ( n % 2 == 0 )`
		`}`

Examples

Input function

```cpp
// C++
int nthTerm ( int n ) {
  return ( n*n ) + ( n*n*n ) ;
}
```

```java
// Java
static int divisorSum(int n){
  int sum = 0 ;
  for(int i=1; i<=n; ++i )
    sum += ( n / i ) * i;
  return sum ;
}
```

```java
// Java
static boolean isDivBy9(int n){
  if ( n == 0  n == 9 )
    return true;
  if ( n < 9 )
    return false;
  return isDivBy9(
        (int)(n >> 3) - (int)(n & 7)
        );
}
```

TransCoder

```rust
// Rust
fn nth_term (n: usize )-> usize {
    ( n*n ) + ( n*n*n )
}
```

```go
// Go
func divisorSum(n int) (int, int) {
 sum := 0
 for i := 1 ; i <= n ; i ++ {
   sum += ( n / i ) * i
 }
 return sum , n
 }
```

```go
// Go
func IsDivBy9 ( n int ) bool {
 if n == 0  n == 9 {
   return true
 }
 if n < 9 {
   return false
 }
 return IsDivBy9 (int(n) > 3 )
           - int(n & 7)
 }
```

TransCoder-IR

```rust
// Rust
fn nth_term ( n : i32 ) -> i32
    ( n*n ) + ( n*n*n )
}
```

```go
// Go
func divisorSum ( n int ) int {
 sum := 0
 for i := 1 ; i <= n ; i ++ {
   sum += ( n / i ) * i
 }
 return sum
}
```

```go
// Go
func IsDivBy9 ( n int ) bool {
 if n == 0  n == 9 {
   return true
 }
 if n < 9 {
  return false
 }
 return IsDivBy9 (
        int(n >> 3) - int(n & 7)}
        )
}
```