Records in IML

Compilerbau HS 2015, Team BB
Team: Livio Bieri, Raphael Brunner

Zwischenbericht 1

Abstract

Zwischenbericht im Modul Compilerbau. Das Dokument beschreibt die Erweiterung **Records in IML** die wir im Rahmen des Moduls vornehmen.

Beschreibung der Erweiterung

Die Erweiterung soll sogenannte **Records** (auch bekannt als *struct* oder *compound data*)[1] zur Verfügung stellen. Ein Record soll dabei als neuer Datentyp zur Verfügung stehen. Er soll beliebig viele Felder beinhalten kann. Felder können vom Datentyp Integer oder Boolean sein. Definierte Records sind im ganzen Programm verfügbar *(in global zu definieren).*

Eine **Deklaration** in IML sieht wie folgt aus:

- `var example: record(x: int64, b: boolean)`

Der **Zugriff** ist wie folgt möglich:

- `debugout example.x`
- Die Felder können vom Datentyp `boolean` oder `int64` sein. *Nested Records sind nicht möglich.*

[1] https://en.wikipedia.org/wiki/Record_(computer_science)

Beispiel

```
program prog
global
    var position: record(x: int64, y: int64);
    const professor: record(id: int64, level: int64)
do
    // all fields must be initialised!
    position(x init := 4, y init := 5);
    professor(id init := 1007, level: 19);

    // fields can be changed
    debugin position.y;
    position.x := 42;

    // field 'id' is const => can not be changed
    // professor.id := 423;

    offsetInY init := 5;
    position.y := position.y + offsetInY

endprogram
```

Funktionalität und Typeinschränkung

Deklaration des Record in `global`

Die Deklaration eines Records muss im `global` vorgenommen werden:

```
...
global
    var position: record(x: int64, y: int64);
    const professor: record(id: int64, level: int64)
do
...
```

Eindeutigkeit des Record Identifier

Die Deklaration (der *Identifier*) eines Records muss eindeutig sein:

```
var position: record(x: int64, y: int64);
const position: record(z: int64, u: int64) // Fehler
      ^^^^^^^^
```

Eindeutigkeit der Record Felder Identifier

Die Deklaration eines Record Felds muss eindeutig sein:

```
var position: record(x: int64, x: int64) // Fehler
                               ^
```

Felder Eindeutigkeit muss aber nur innerhalb *eines Records gegeben sein:*

```
var positionXY: record(x: int64, y: int64);
var positionXYZ: record(x: int64, y: int64, z: int64)
```

Typenchecking (`bool`, `int64`)

Der zugewiesene Wert muss vom Typ sein, der in der Deklaration angegeben wurde (`bool` oder `int64`):

```
var point: record(x: int64, y: int64);

point.x := true; // Fehler
           ^^^^
```

Zugriff auf undefiniert Felder

Der Zugriff auf Felder die nicht definiert wurden ist nicht möglich:

```
var point: record(x: int64, y: int64);

point.z = 42; // Fehler
      ^
```

Unterstützung von `CHANGEMODE` in Records

Records unterstützen `CHANGEMODE` (`var`, `const`):

- `CHANGEMODE` ist optional.
- Falls nicht angegeben wird `var` verwendet.

```
point: record(x: int64, y: int64)
```

Wird interpretiert als:

```
var point: record(x: int64, y: int64)
```

Felder unterstützen *kein* `CHANGEMODE`:

```
point: record(const x: int64, y: int64) // Fehler
              ^^^^^
```

Operationen auf Records:

Records selbst haben *keine Operationen. Folgendes ist also nicht möglich / wird nicht unterstützt:*

```
pointZero: record(var x: int64, var y: int64);
pointOne: record(var x: int64, var y: int64)
...

pointZero = pointZero + pointOne // Fehler
                      ^
```

Vergleich mit anderen Sprachen

Wir haben uns unterschiedliche Lösungsansätze angeschaut. Dazu haben wir uns vor allem angeschaut, was andere Sprachen konkret machen:

Haskell

Deklaration:

```
data vector = vector (
    x..Int, y..Int, z..Int)
```

Initialisierung:

```
v1 = vector 5 6 7
```

Pascal

Deklaration:

```
type TVector = record
    x : Integer ;   y : Integer ;   z : Integer ;
end ;
```

Initialisierung:

```
var v1 : TVector
begin
    v1.x := 42;
    v1.y := 50;
    v1.z := 20;
end ;
```

C

Deklaration:

```
struct vector {
    int x;
    int y;
    int z;
};
```

Initialisierung:

```
vector v1;
v1.x = 42;
v1.y = 50;
v1.z = 20;
```

IML

Deklaration:

```
var v1: record(x: int64, y: int64, z: int64)
```

Initialisierung:

```
v1(x init := 42, y init := 42, z init := 42)
```

Einfluss auf unsere Lösung:

- Unser Ziel war es eine IML-ähnliche Syntax beizubehalten.
- Unsere Implementation orientiert sich an der Pascal Implementation.
- Wir fanden eine einfache Initialisierung wichtig *(in einer Zeile).*

Lexikalische und Grammatikalische Syntax

```
!!!
```

Sonstiges

- Sourcecode & Dokumentation: https://github.com/livioso/cpib
- Arbeitsteilung: Wir haben die Arbeiten wie folgt im Team verteilt. *Bieri: Scanner, Zwischenbericht / Brunner: Grammatik (SML), Zwischenbericht.*