Undestand の解析

Understand の C 言語で抽出できない依存関係について、サンプルコードを用いて説明します。

確認バージョン

- Understand 3.0 (Build 640)
- Understand 3.1 (Build 700)
- Understand 4.0 (Build 788)

抽出できない依存関係

Understand の C 言語の解析（Fuzzy/Strict）で、抽出できない依存関係には、以下があります。

- ポインタ関連
  - 関数ポインタによる関数の呼び出し
  - 関数テーブル（ジャンプテーブル）
  - ポインタ変数のアドレス操作

    上記例以外でも、ポインタ経由の操作については依存関係を解析できません（例:コールバック関数）
- 同名定義

関数ポインタによる関数の呼び出し

サンプルコード

```
01  #include <stdio.h>
02
03  void (*pf)(char*);
04
05  void func(char* c) {
06    printf("%s¥n", c);
07  }
08
09  void foo1() {
10    pf = func;
11  }
12
13  void foo2() {
14    pf("hello");
15  }
```

```
16
17  void foo3() {
18    func("world");
19  }
20
21  int main() {
22    foo1();
23    foo2();
24    foo3();
25    return 0;
26  }
```

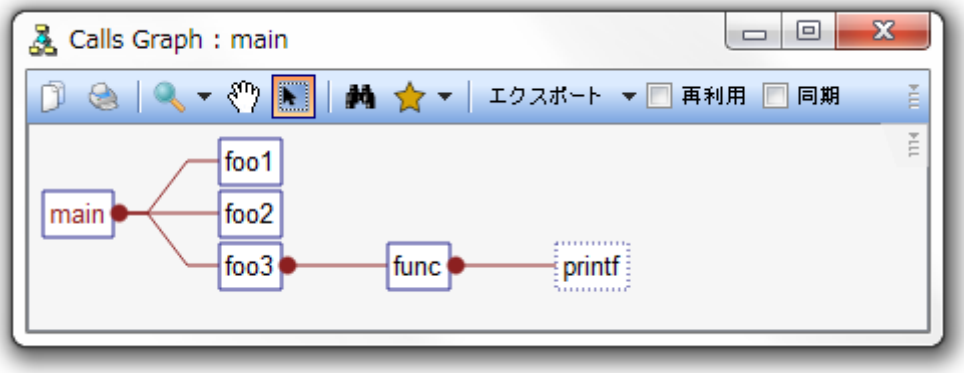
[Fuzzy/Strict 解析]

main 関数の Calls Graph

Calls Graph のコンテキストメニュー[Function Pointer]を[On]に切り替えた場合、情報ブラウザーの References で表示される「call-deref」の関数ポインタのアドレス参照や「pointer」で表示される関数への呼び出しが表示されます。しかし、関数ポインタ(pf)に関数アドレスを代入する処理(10 行目)も、関数呼び出しと見なされ、表示されます。

- コンテキストメニュー[Function Pointer]-[Off]の場合：

  実際の関数呼び出しのみ表示します。

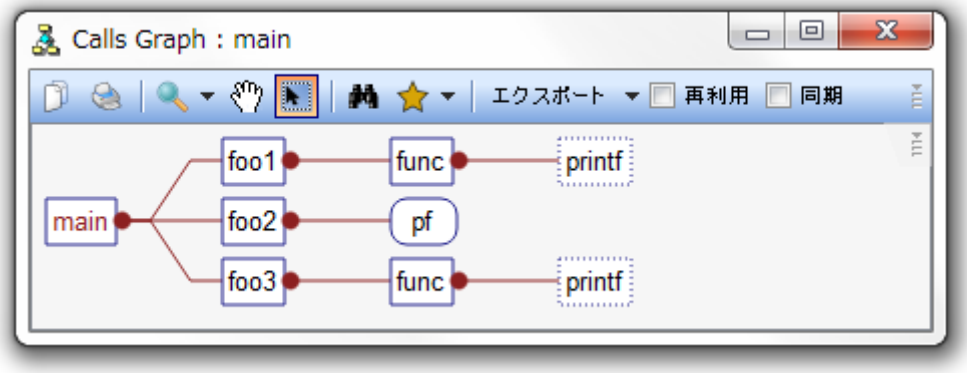


- コンテキストメニュー[Function Pointer]-[On]の場合：

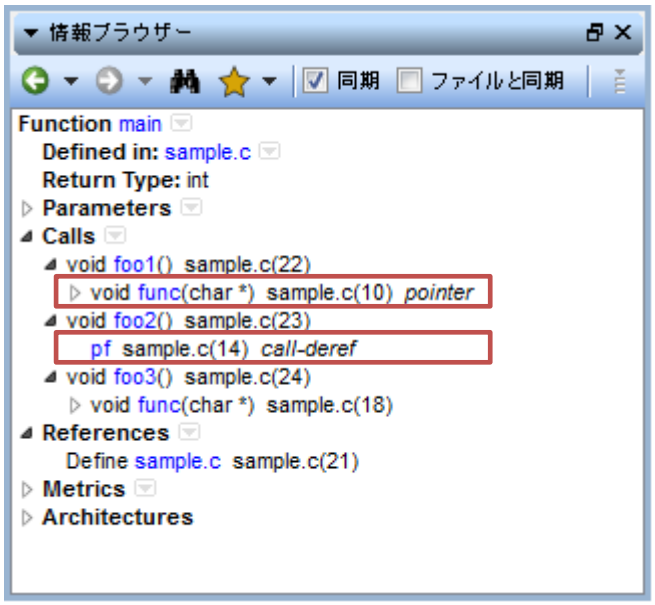
  情報ブラウザーの Calls セクション内の

  - 「void func(char *) sample.c(10) *pointer*」で示されている、関数へのポインタ宣言
  - 「pf sample.c(14) *call-deref*」で示されている、デリファレンス（参照外し）による関数呼び出し

  が表示されますが、デリファレンスで呼び出される関数の実体（この場合 func）を解析することができません。

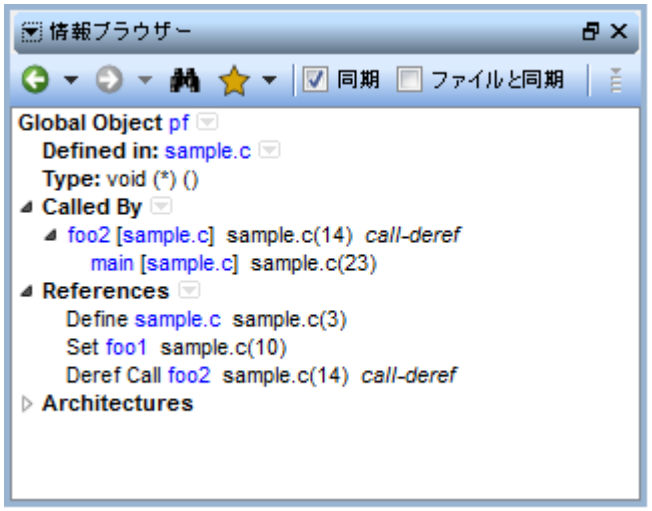


main 関数の情報ブラウザーの表示



関数ポインタ(pf)の情報ブラウザーの表示

pf の実体が func()であることが解析できません。



関数テーブル（ジャンプテーブル）

サンプルコード

```
01  #include <stdio.h>
02
03  void Event01(){ printf("Event01¥n"); };
04  void Event02(){ printf("Event02¥n"); };
05  void Event03(){ printf("Event03¥n"); };
06
07  void (*funcTable[])() =
08  {
09    Event01,
10    Event02,
11    Event03
12  };
13
14  int foo()
15  {
16    int i;
17    for ( i = 0; i < 3; i++ ){
18      funcTable[i](1);
19    }
20    return 0;
21  }
```

[Strict 解析]

funcTable のデリファレンスによる関数呼び出しとして解析されますが、関数の実体(Event01, Event02, Event03)を解析できません。

foo 関数の Calls Graph

funcTable の情報ブラウザーの表示

[Fuzzy 解析]

デリファレンスによる関数呼び出しとしても表示されません。funcTable 配列オブジェクトの Useとして処理されます。

ポインタ変数のアドレス操作

サンプルコード

```
01  #include <stdio.h>
02
03  typedef struct {
04    int x;
05    int y;
06    int z;
07  } STRC;
08
09  int main()
10  {
11    STRC data;
12    STRC *dataPtr;
13
14    data.x = 1;
15    data.y = 2;
16    data.z = 3;
17
18    dataPtr = &data;
19
20    dataPtr->x = data.z;
21    printf("%d, %d", dataPtr->x, dataPtr->y);
22
23    return 0;
24  }
```

[Fuzzy/Strict 解析]

ポインタ変数にアドレスを渡して、そのポインタ先でアドレス操作した場合、その実体との参照関係は表示できません。上記の例では、ポインタ変数(dataPtr)から、実体(data)のメンバーへのアクセスは(20 行目を data.x への Set, 21 行目を data.y への Use として)検出できません。

構造体の実体(data)の情報ブラウザーの表示

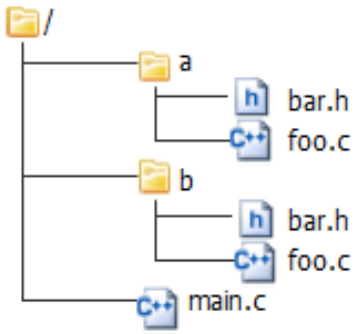
同名・多重定義

C 言語の場合、実際のコンパイル単位では、同名の変数・関数は多重定義することはできません。Understand では、同名・多重定義があった場合もエラーを発生しないで解析可能ですが、同名の変数や関数への参照は、正しい解析結果が得られない場合があります。

対応方法

プロジェクトから同名定義を取り除いたプロジェクト単位で解析するようにしてください。

サンプルコードのディレクトリ構造



サンプルコード(a/bar.h)

```
01  #ifndef BAR_H
02  #define BAR_H
03
04  struct AAA {
05   int a;
06   int b;
07  };
08
09  struct AAA aaa;
10  int gnum = 100;
11  void func(int);
12
13  #endif BAR_H
```

サンプルコード(a/foo.c)

```
01  #include <stdio.h>
02  #include "bar.h"
03
04  void func(int num)
05  {
06    aaa.a = 1;
07    aaa.b = 2;
08    gnum = 100;
09    printf("a:%d", num);
10  }
```

サンプルコード(b/bar.h)

```
01  struct AAA {
02   int a;
03   int b;
04  };
05
06  struct AAA aaa;
07  int gnum = 200;
08  void func(int);
```

サンプルコード(b/foo.c)

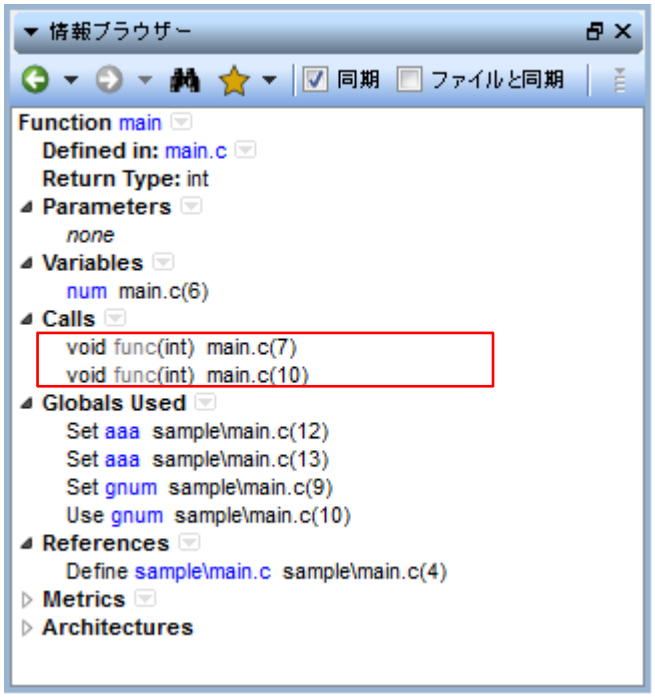
```
01  #include <stdio.h>
02  #include "bar.h"
03
04  void func(int num)
05  {
06    aaa.a = 10;
07    aaa.b = 20;
08    gnum = 200;
09    printf("b:%d", num);
10  }
```

サンプルコード(main.c)

```
01  #include <stdio.h>
02  #include "bar.h"
03
04  int main()
05  {
06    int num = 10;
07    func(num);
08
09    gnum = 300;
10    func(gnum);
11
12    aaa.a = 10;
13    aaa.b = 20;
14
15    return 0;
16  }
```

[Fuzzy 解析]

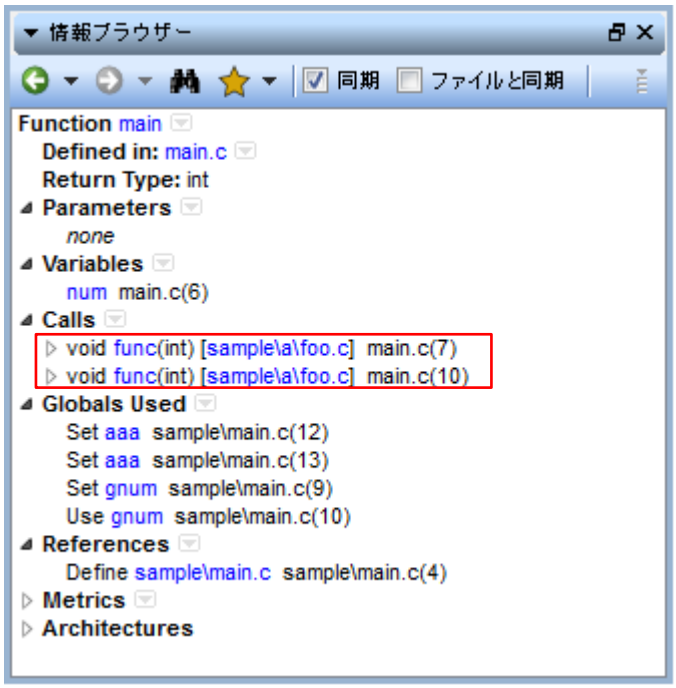
main 関数の情報ブラウザーの表示



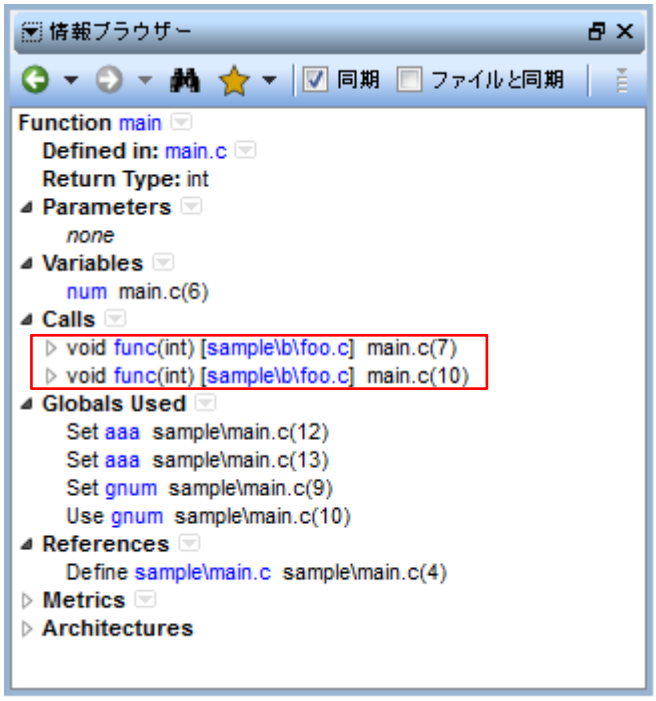
同名の関数(func)が存在することで、定義元が解決できず、未解決のエンティティ（グレーアウト）として処理されています。

[Strict 解析]

main 関数の情報ブラウザーの表示



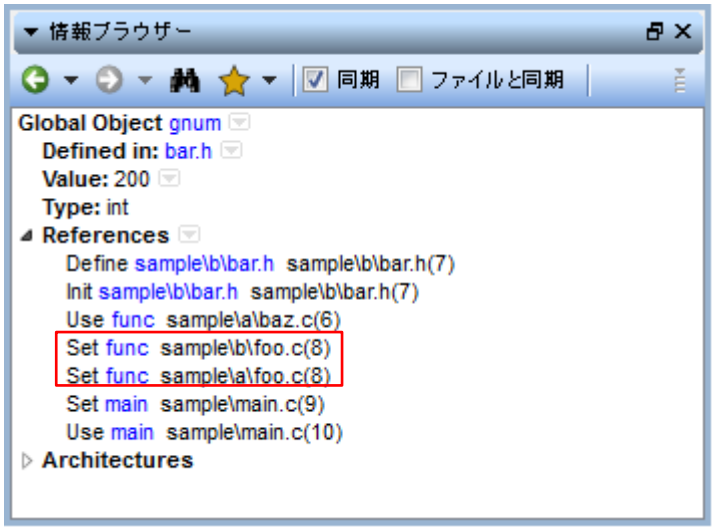
同名の関数(func)は、a/foo.c で定義されたものとして処理されていますが、複数の同名の要素がある場合、その定義元はランダムで参照されます。そのため、下記のように、b/foo.c の func 関数への呼び出しとして解析される場合もあります。



Understand のオーバーライドの設定機能で、ファイルやディレクトリに対して、例えばインクルードディレクトリを個別に設定することができます。しかし、この設定をおこなっても、エンティティの種類によっては解析結果に反映されない場合があります。

[Fuzzy/Strict 解析]

グローバル変数(b/bar.h の gnum)の情報ブラウザーの表示



b/bar.h で宣言されているグローバル変数 gnum が、a/foo.c と b/foo.c の func 関数の両方から参照関係があると解析されています。