Automatic Source Code Reduction

Jiří Diviš, Ondřej Bojar

This work has been supported by the grants Euro-MatrixPlus (FP7-ICT-2007-3-231720 of the EU and 7E09003 of the Czech Republic) and MSM 0021620838.

Obsah

- Programovací jazyk Mercury
- Motivace
- Reprezentace programu
- Redukce podrobně
- Rozhraní pro přenesení výsledků redukce zpět do zdrojáku.

Programovací jazyk Mercury

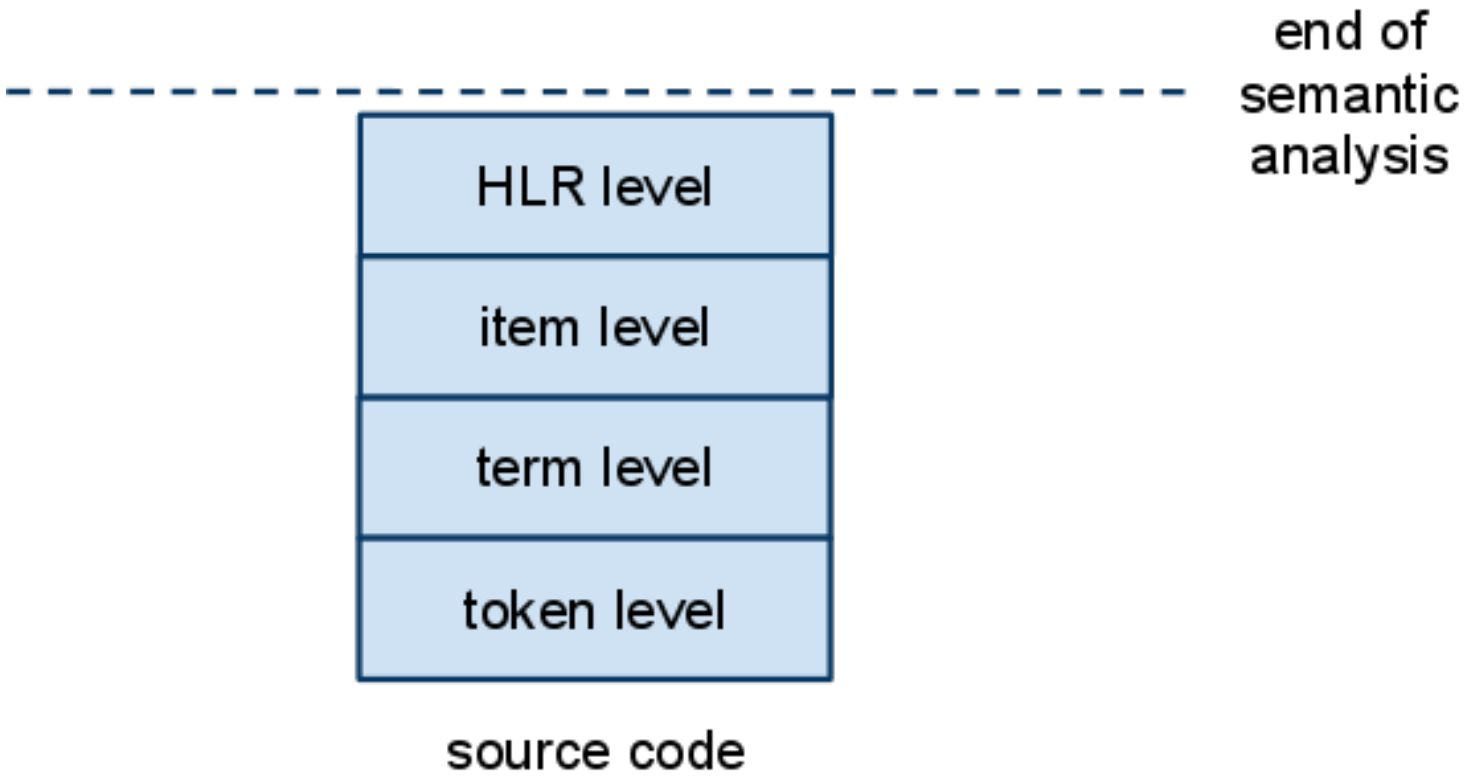
- V zásadě rozšíření Prologu
  - Deklarace a definice typů proměných
  - Deklarace směru toku dat (tzv. mode).
  - Informace zda prdikát smí selhat, nebo vrátit více výsledků
- kompiluje také do C nebo Javy
- Foreign Language Interface
- Oproti Prologům běží výrazně rychleji a snadněji se v něm odlaďují programy

Motivace, Reductor a jeho redukce

- Funkcionalita:
  - Statická redukce, dynamická redukce
  - Zachovává původní strukturu kódu
    - Formátování
    - Jména proměných
    - ap.
- Účel:
  - Hlavně porozumění nebo znovupoužití rozsáhlých zdrojáků
  - Pripřípadně “urychlení” kompilace, nebo “zmenšení” binárky

Levels of representation

- Melbourne Mercury Compiler (MMC)
- Kód z MMC, přizpůsobený pro Reductor



Dynamická redukce

- Mercury Deep Profiler
  - Získání informací o volání predikatů
    - krabičkový model + vyjímky
- Kompilace az po HLR
  - Mode reordering
  - Předpoklad: strict sequential operational semantics
- Přenos informací o volání do HLR
- Redukce
- Predaní výstupu I/O modulu

Dynamická redukce – příklad

–

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, append(B0, H, B), split(P, S, A, B0))
;
    (P<H, append(A0, H, A), split(P, S, A0, B)).
```

Dynamická redukce – mode reordering

–

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, split(P, S, A, B0), append(B0, H, B))
;
    (P<H, split(P, S, A0, B), append(A0, H, A)).
```

Dynamická redukce – sběr informací

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, split(P, S, A, B0), append(B0, H, B))
;
    (P<H, split(P, S, A0, B), append(A0, H, A)).
```

Dynamická redukce – sběr informací

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, split(P, S, A, B0), append(B0, H, B))
;
    (P<H, split(P, S, A0, B), append(A0, H, A)).
```

Dynamická redukce – sběr informací

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, split(P, S, A, B0), append(B0, H, B))
;
    (P<H, split(P, S, A0, B), append(A0, H, A)).
```

Dynamická redukce – označení

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, split(P, S, A, B0), append(B0, H, B))
;
    (P<H, split(P, S, A0, B), append(A0, H, A)).
```

Dynamická redukce – označení

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, append(B0, H, B), split(P, S, A, B0))
;
    (P<H, append(A0, H, A), split(P, S, A0, B)).
```

Dynamická redukce – označení

Vstup: 2 a [1,2]

```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, append(B0, H, B), split(P, S, A, B0))
;
    (P<H, append(A0, H, A), split(P, S, A0, B)).
```

Dynamická redukce – výsledek

Vstup: 2 a [1,2]

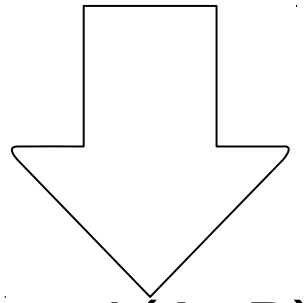
```
split(P, [], []).

split(P, [H|S], A, B):-
    (P>H, throw(“red'd”)/*append(B0, H, B),split(P, S, A, B0)*/)
;
    (P<H, append(A0, H, A), split(P, S, A0, B)).
```

Rozhraní pro přenesení redukcí zpět do zdrojáků

- Interface pro zápis změn redukcí (I/O interface)
- Prijma změny typu:
  - Nahraď term podtermem
  - Přepiš term řetězcem
- Rozhraní hledá znakové pozice ve zdrojáku
- Priklad: term ↔ token
- Takto čistě jen term ↔ token
  - Další vrstvy jdou podobně též
    - Hezké, ale pracné
    - Pro implementované redukce zbytečné

token level <-> term level

```
term1(A, B):-
    term2(cons(1,A),Y),
    term4(1,B).
```

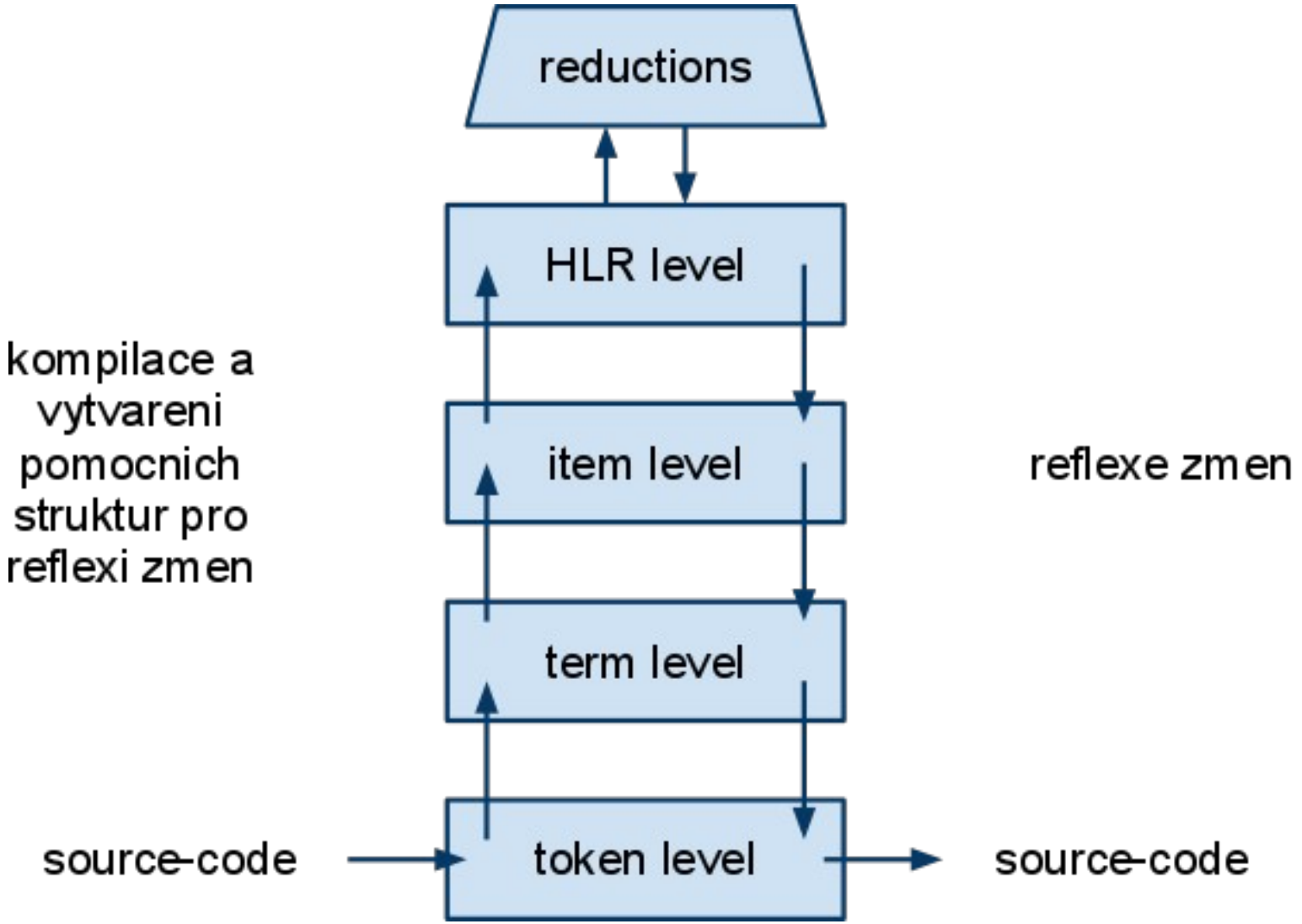


```
term1(A, B):-
    term2(A,Y),
    throw,
    term4(Y,B).
```

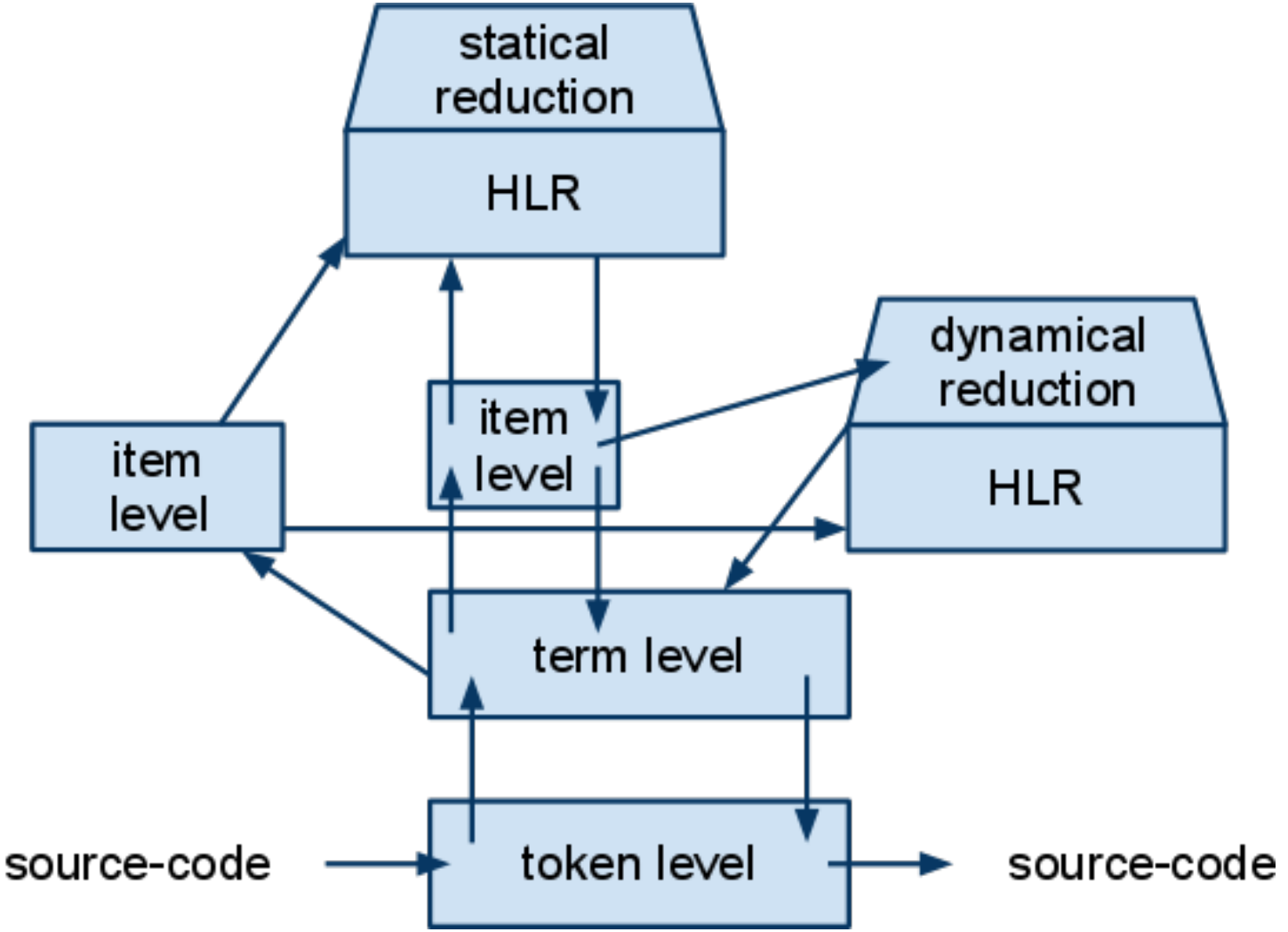
```
 `:-`
term(A,B)
`,`
   term2
      cons
         1
         A
      Y


term4
      1
      A
```

```
no_lvl_change |`:-`|
  no_change |term1|
  no_lvl_change |`,`|
     no_lvl_change |term2|
        subst_del |cons|
           no_change |1|
           orig([])  |A|
        no_change |Y|
     subst_ins
        "<NL><tab>throw<NL>"
        no_lvl_change |term4|
           replace("Y") |1|
           no_change    |A|
        ""
```

Ideální rozhraní



Rozhraní – realita



Závěrem

- Bohužel není vůbec snadné ověřit, jak moc se Reductor skutečně hodí ke svému hlavnímu účelu
  - Není na čem to vyzkoušet...
- Jinak k “urychlení” kompilace, nebo “zmenšení” binárky se samozřejmě hodí

Statická redukce

- jde implementovat jako pruchod grafem.
- Problemy prinasi polymorfizmus:
- existencial types
- typeclasses
- lze si predstavovat jako virtualni metody z OOP