Efficient Interpretation of I-MSOS

via an Intermediate Language

L. Thomas van Binsbergen[1]
ltvanbinsbergen@acm.org
Neil Sculthorpe[2] Adrian Johnstone[1] Elizabeth Scott[1]

[1]Royal Holloway, University of London

[2]Nottingham Trent University

September 2, 2016

Section 1

PLanCompS

The PLanCompS Approach

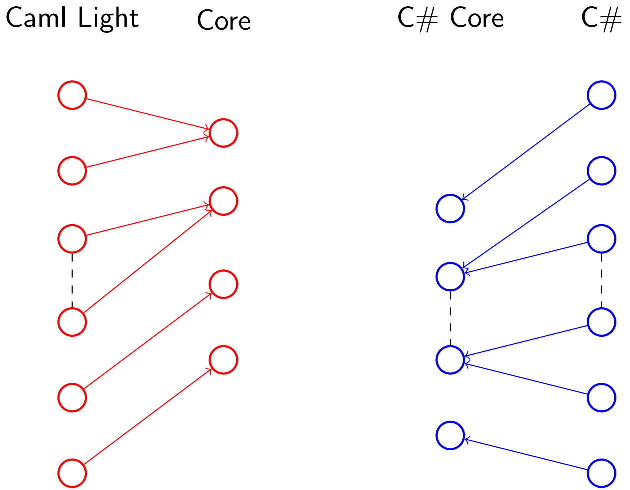
PLanCompS project (2011-2015)

- Swansea University
- Royal Holloway, University of London
- http://plancomps.org

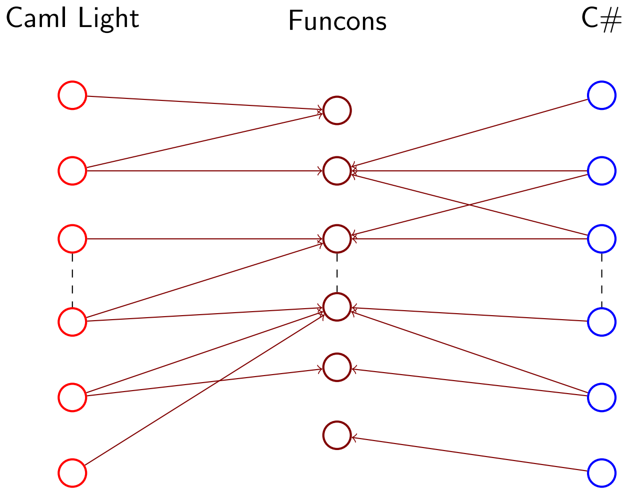
Approach

- Component based approach towards formal semantics.
- Highly reusable, fundamental constructs: *funcons*.
- A language is defined formally via a translation to funcons.
- Each funcon has a formal definition in I-MSOS.

Reusable Components: Funcons



Reusable Components: Funcons



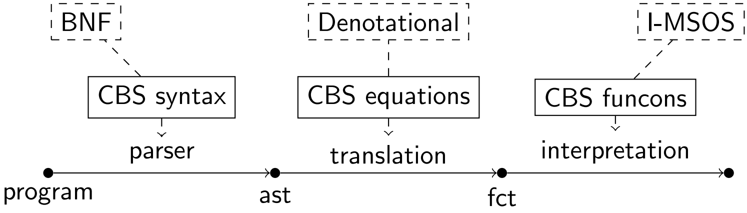


Figure : PLanCompS: generate interpreters from reusable specification.

Section 2

I-MSOS Specification

Relations

In this talk we consider relations

$$R(X, Y, Z)$$

for reasoning about program execution

- Relations are defined via logical inference rules
- Relations are computed by semantic functions

How to generate efficient semantic functions?

Relations and Functions

- We can define a function $\overrightarrow{F}$ for computing a relation $\overrightarrow{R}$:

$$\begin{aligned}
\overrightarrow{R}(X,Y,Z) &\Leftrightarrow & \mathit{true} = &\ \overrightarrow{F}(X,Y,Z) \\
\overrightarrow{R}(X,Y,Z) &\Leftrightarrow & Z \in &\ \overrightarrow{F}(X,Y) \\
\overrightarrow{R}(X,Y,Z) &\Leftrightarrow & (Y,Z) \in &\ \overrightarrow{F}(X)
\end{aligned}$$

- or approximating a relation:

$$\overrightarrow{R}(X,Y,Z) \ \Leftarrow\ (Y,Z) = \overrightarrow{F}(X)$$

- based on the inference rules defining the relation $\overrightarrow{R}$
- Above X is *input* of $\overrightarrow{F}$; Y and Z are *output* of $\overrightarrow{F}$

Relations modeling program execution

- Terms (computations and values) represent program fragments

$$\vec{R}(\mathit{plus}(1,2),3,\sigma) \quad \textit{is abbreviated by} \quad \sigma \vdash \mathit{plus}(1,2) \rightarrow 3$$

with σ some arbitrary environment

- Auxiliary semantic entities like *environment* σ provide context

Program semantics

- Relation $\vec{R}$ describes transitions on terms
- Semantic function $\vec{F}$ 'performs' a transition

$$\sigma \vdash \mathit{plus}(1,2) \rightarrow 3 \quad \Leftarrow \quad 3 = \vec{F}(\mathit{plus}(1,2),\sigma)$$

Terminology

- The following are *judgements* about a relation

$$\vec{R}(X, Y, Z)$$

$$\sigma \vdash plus(1,2) \rightarrow 3$$

- About $\vec{F}(plus(1,2), \sigma) = 3$, we say

input term	$plus(1,2)$
input arguments	$1,2$
output term	3
additional input	σ
additional output	$\ldots$

$$\begin{array}{rcll} E & : & \mathit{expr} & ::= \quad \textbf{plus}\ E\ E \mid \textbf{leq}\ E\ E \mid R \mid V \\ V & : & \mathit{value} & ::= \quad \textbf{false} \mid \textbf{true} \mid I \\ I & : & \mathit{integer} & ::= \quad \ldots \\ R & : & \mathit{reference} & ::= \quad \ldots \end{array}$$

Figure : Concrete syntax for a small EXPR language.

$$E \quad : \quad \mathit{expr} \quad ::= \quad \mathit{plus}(E_1, E_2) \mid \mathit{leq}(E_1, E_2) \mid \mathit{var}(R) \mid V$$

Figure : Term representation of expressions.

$$\frac{\sigma \vdash E_1 \to I_1 \qquad \sigma \vdash E_2 \to I_2}{\sigma \vdash plus(E_1, E_2) \to I_3} (I_3 = I_1 + I_2)$$

Figure : SOS rules for expressions.

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash plus(E_1, E_2) \rightarrow I_3}(I_3 = I_1 + I_2)$$

Figure : SOS rules for expressions.

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash leq(E_1, E_2) \rightarrow \textbf{true}}\,(I_1 \leqslant I_2)$$

Figure : SOS rules for expressions.

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash leq(E_1, E_2) \rightarrow \mathbf{true}}\,(I_1 \leqslant I_2)$$

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash leq(E_1, E_2) \rightarrow \mathbf{false}}\,(I_1 \nleqslant I_2)$$

Figure : SOS rules for expressions.

$$\frac{}{\sigma \vdash var(R) \rightarrow V}\,(V = \sigma(R))$$

Figure : SOS rules for expressions.

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash plus(E_1, E_2) \rightarrow I_3}\,(I_3 = I_1 + I_2)$$

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash leq(E_1, E_2) \rightarrow \mathbf{true}}\,(I_1 \leqslant I_2)$$

$$\frac{\sigma \vdash E_1 \rightarrow I_1 \qquad \sigma \vdash E_2 \rightarrow I_2}{\sigma \vdash leq(E_1, E_2) \rightarrow \mathbf{false}}\,(I_1 \not\leqslant I_2)$$

$$\frac{}{\sigma \vdash var(R) \rightarrow V}\,^{(V = \sigma(R))}$$

Figure : SOS rules for expressions.

$$\frac{E_1 \rightarrow I_1 \qquad E_2 \rightarrow I_2}{plus(E_1, E_2) \rightarrow I_3}(I_3 = I_1 + I_2)$$

$$\frac{E_1 \rightarrow I_1 \qquad E_2 \rightarrow I_2}{leq(E_1, E_2) \rightarrow \mathbf{true}}(I_1 \leqslant I_2)$$

$$\frac{E_1 \rightarrow I_1 \qquad E_2 \rightarrow I_2}{leq(E_1, E_2) \rightarrow \mathbf{false}}(I_1 \nleqslant I_2)$$

$$\frac{}{\mathsf{env}(\sigma) \vdash var(R) \rightarrow V}^{(V = \sigma(R))}$$

Figure : I-MSOS rules for expressions.

Modularity and SOS

- Additional semantic entities requires *changing the relation*
- For example, a quaternary evaluation relation:

$$\overrightarrow{R}(T_0, T_0', \sigma, \sigma') \quad \textit{abbreviated as} \quad \langle T_0, \sigma \rangle \rightarrow \langle T_0', \sigma' \rangle$$

Modularity and I-MSOS

- *All* I-MSOS relations are senary:

$$\overrightarrow{R}(T_0, T_1, E^{\downarrow}, E_0^{\updownarrow}, E_1^{\updownarrow}, E^{\uparrow})$$

abbreviated by

$$E^{\downarrow} \vdash \langle T_0, E_0^{\updownarrow} \rangle \xrightarrow{E^{\uparrow}} \langle T_1, E_1^{\updownarrow} \rangle$$

where $E^{\downarrow}$, $E_i^{\updownarrow}$ and $E^{\uparrow}$ are *records* storing semantic entities

- *Read-only* ($\downarrow$) entities provide additional input
- *Write-only* ($\uparrow$) entities provide additional output
- *Read-write* ($\updownarrow$) entities provide additional input and output

$$E \quad : \quad expr \quad ::= \quad \textbf{print}\ E \mid R{+}{+} \mid \ldots$$

Figure : Additional concrete syntax for EXPR.

$$E \quad : \quad expr \quad ::= \quad print(E) \mid inc(R) \mid \ldots$$

Figure : Term representation of additional syntax

$$\frac{}{\langle var(R), \mathsf{env}(\sigma)\rangle \rightarrow \langle V, \mathsf{env}(\sigma)\rangle}\,(V{=}\sigma(R))$$

Figure : New version of variable reference

$$\frac{}{\langle var(R), \mathsf{env}(\sigma)\rangle \rightarrow \langle V, \mathsf{env}(\sigma)\rangle}\,(V{=}\sigma(R))$$

Figure : New version of variable reference

$$\frac{}{\langle inc(R), \mathsf{env}(\sigma)\rangle \rightarrow \langle V_1, \mathsf{env}(\sigma[R \mapsto V_2])\rangle}\,(V_1{=}\sigma(R), V_2{=}V_1{+}1)$$

Figure : I-MSOS rules for additional expressions.

$$\frac{}{\langle var(R), \mathsf{env}(\sigma)\rangle \rightarrow \langle V, \mathsf{env}(\sigma)\rangle}(V{=}\sigma(R))$$

Figure : New version of variable reference

$$\frac{E \xrightarrow{\mathrm{out}(\alpha)} V}{print(E) \xrightarrow{\mathrm{out}(\alpha +\!\!+ [V])} V}$$

Figure : I-MSOS rules for additional expressions.

$$\frac{}{\langle var(R), \mathrm{env}(\sigma)\rangle \rightarrow \langle V, \mathrm{env}(\sigma)\rangle}\,(V{=}\sigma(R))$$

Figure : New version of variable reference

$$\frac{}{\langle inc(R), \mathrm{env}(\sigma)\rangle \rightarrow \langle V_1, \mathrm{env}(\sigma[R \mapsto V_2])\rangle}\,(V_1{=}\sigma(R), V_2{=}V_1{+}1)$$

$$\frac{E \xrightarrow{\mathrm{out}(\alpha)} V}{print(E) \xrightarrow{\mathrm{out}(\alpha + [V])} V}$$

Figure : I-MSOS rules for additional expressions.

$$\frac{\mathsf{ro\text{-}ent}(\rho) \vdash t_1 \rightarrow_1 p_1 \quad \ldots \quad \mathsf{ro\text{-}ent}(\rho) \vdash t_n \rightarrow_n p_n}{\mathsf{ro\text{-}ent}(\rho) \vdash t_0 \rightarrow_0 p_0}$$

Figure : Rules for implicit propagation of unmentioned entities.

$$\frac{\text{ro-ent}(\rho) \vdash t_1 \rightarrow_1 p_1 \quad \ldots \quad \text{ro-ent}(\rho) \vdash t_n \rightarrow_n p_n}{\text{ro-ent}(\rho) \vdash t_0 \rightarrow_0 p_0}$$

$$\frac{\langle t_1, \text{rw-ent}(\sigma_0)\rangle \rightarrow_1 \langle p_1, \text{rw-ent}(\sigma_1)\rangle \quad \ldots \quad \langle t_n, \text{rw-ent}(\sigma_{n-1})\rangle \rightarrow_n \langle p_n, \text{rw-ent}(\sigma_n)\rangle}{\langle t_0, \text{rw-ent}(\sigma_0)\rangle \rightarrow_0 \langle p_0, \text{rw-ent}(\sigma_n)\rangle}$$

Figure : Rules for implicit propagation of unmentioned entities.

$$\frac{\text{ro-ent}(\rho) \vdash t_1 \rightarrow_1 p_1 \quad \dots \quad \text{ro-ent}(\rho) \vdash t_n \rightarrow_n p_n}{\text{ro-ent}(\rho) \vdash t_0 \rightarrow_0 p_0}$$

$$\frac{\langle t_1, \text{rw-ent}(\sigma_0)\rangle \rightarrow_1 \langle p_1, \text{rw-ent}(\sigma_1)\rangle \quad \dots \quad \langle t_n, \text{rw-ent}(\sigma_{n-1})\rangle \rightarrow_n \langle p_n, \text{rw-ent}(\sigma_n)\rangle}{\langle t_0, \text{rw-ent}(\sigma_0)\rangle \rightarrow_0 \langle p_0, \text{rw-ent}(\sigma_n)\rangle}$$

$$\frac{t_1 \xrightarrow{\text{wo-ent}(\alpha_1)}_1 p_1 \quad \dots \quad t_n \xrightarrow{\text{wo-ent}(\alpha_n)}_n p_n}{t_0 \xrightarrow{\text{wo-ent}(\alpha_1 +\!\!+ \dots +\!\!+ \alpha_n)}_0 p_0}$$

Figure : Rules for implicit propagation of unmentioned entities.

Section 3

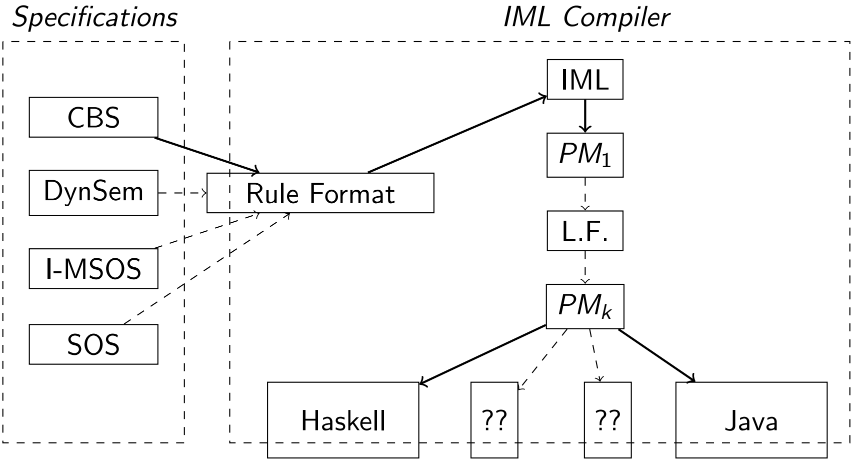
I-MSOS intermediate language (IML)

Aim

Generate efficient interpreters from I-MSOS specifications

Approach

- The *IML Rule Format* generalises I-MSOS inference rules
- IML rules generate low-level *IML procedures*
- IML procedures are relatively easy to optimise:
  - Left-factoring
  - Statement reordering
  - Common subexpression elimination
  - ...


Specifications
IML Compiler
CBS
DynSem
I-MSOS
SOS
Rule Format
IML
PM_1
L.F.
PM_k
Haskell
??
??
Java

Value operations

- Inference rules specify the semantics of computations, however:

  values, and operations on values, are assumed

  - integer addition $I_3 = I_1 + I_2$
  - integer comparison $I_1 \leqslant I_2$ and $I_1 \not\leqslant I_2$
  - map lookup $V = \sigma(R)$
  - map extension/insertion $\sigma[R \mapsto V]$

- Any required operations must be available to an interpreter

Operator-expressions

- An operator-expression is either:
  - A value v
  - An operation o applied to operator-expressions: $o(e_1, \ldots, e_n)$

Operator-expressions

- An operator-expression is either:
  - A value v
  - An operation o applied to operator-expressions: $o(e_1, \ldots, e_n)$

IML Rule Format

- Given distinct index sets I and J, with $0 \notin I$:

$$\frac{\{R_i(t_i, p_i, E_i^{\downarrow}, E_{0,i}^{\updownarrow}, E_{1,i}^{\updownarrow}, E_i^{\uparrow}) : i \in I\}}{R_0(f(w_1, \ldots, w_n), t, E_0^{\downarrow}, E_{0,0}^{\updownarrow}, E_{1,0}^{\updownarrow}, E_0^{\uparrow})}(\{e_j \triangleright p_j : j \in J\})$$

- Recall $R(T_0, T_1, E^{\downarrow}, E_0^{\updownarrow}, E_1^{\updownarrow}, E^{\uparrow})$ is an I-MSOS judgement
- $e_j \triangleright p_j$ denotes a side-condition

IML Rule Format

- Given distinct index sets I and J, with $0 \notin I$:

$$\frac{\{R_i(t_i, p_i, E_i^{\downarrow}, E_{0,i}^{\updownarrow}, E_{1,i}^{\updownarrow}, E_i^{\uparrow}) : i \in I\}}{R_0(f(w_1, \ldots, w_n), t, E_0^{\downarrow}, E_{0,0}^{\updownarrow}, E_{1,0}^{\updownarrow}, E_0^{\uparrow})} (\{e_j \triangleright p_j : j \in J\})$$

- Recall $R(T_0, T_1, E^{\downarrow}, E_0^{\updownarrow}, E_1^{\updownarrow}, E^{\uparrow})$ is an I-MSOS judgement
- $e_j \triangleright p_j$ denotes a side-condition

Example IML Rule

$$\frac{E_1 \rightarrow l_1 \quad E_2 \rightarrow l_2}{leq(E_1, E_2) \rightarrow true} (\texttt{is-leq}(l_1, l_2) \triangleright true)$$

IML Rule Format

- Given distinct index sets I and J, with $0 \notin I$:

$$\frac{\{R_i(t_i, p_i, E_i^{\downarrow}, E_{0,i}^{\updownarrow}, E_{1,i}^{\updownarrow}, E_i^{\uparrow}) : i \in I\}}{R_0(f(w_1, \ldots, w_n), t, E_0^{\downarrow}, E_{0,0}^{\updownarrow}, E_{1,0}^{\updownarrow}, E_0^{\uparrow})}(\{e_j \triangleright p_j : j \in J\})$$

- Recall $R(T_0, T_1, E^{\downarrow}, E_0^{\updownarrow}, E_1^{\updownarrow}, E^{\uparrow})$ is an I-MSOS judgement
- $e_j \triangleright p_j$ denotes a side-condition

Example IML Rule

$$\frac{}{\langle var(R), \mathsf{env}(\sigma)\rangle \rightarrow \langle V, \mathsf{env}(\sigma)\rangle}(\texttt{map-lookup}(R,\sigma) \triangleright V)$$

IML Procedures

- A procedure has *branches*, one for every rule it implements
- A branch is a sequence of statements, ending with
  - A return statement (yielding the output term)
  - Further branches (branches never re-converge)
- Few types of statements, e.g. for:
  - Pattern matching
  - Entity getting and setting
  - Procedure calls
- Statements can fail, causing control to *backtrack* to the last branching point

$$\frac{E_1 \xrightarrow{\mathtt{out}(\alpha)} I_1 \quad E_2 \xrightarrow{\mathtt{out}(\beta)} I_2}{leq(E_1, E_2) \xrightarrow{\mathtt{out}(\alpha +\!\!+ \beta)} true} (\mathtt{is\text{-}leq}(I_1,I_2) \triangleright true)$$

PROCEDURE FOR $\mathbf{leq}^{\rightarrow}$

$pm\text{-}args(E_1, E_2);$
$single(\text{“}\rightarrow\text{”}, E_1, I_1, 1);$
$wo\_get(\mathtt{out}, \alpha, 1);$
$single(\text{“}\rightarrow\text{”}, E_2, I_2, 2);$
$wo\_get(\mathtt{out}, \beta, 2);$
$pm(\mathtt{is\text{-}leq}(I_1, I_2), x_0);$
$pm(x_0, true);$
$wo\_set(\mathtt{out}, \mathtt{list\text{-}append}(\alpha, \beta));$
return *true*

$pm\text{-}args(E_1, E_2);$
$single(\text{“}\rightarrow\text{”}, E_1, I_1, 1);$
$wo\_get(\mathtt{out}, \alpha, 1);$
$single(\text{“}\rightarrow\text{”}, E_2, I_2, 2);$
$wo\_get(\mathtt{out}, \beta, 2);$
$pm(\mathtt{is\text{-}leq}(I_1, I_2), x_0);$
$pm(x_0, false);$
$wo\_set(\mathtt{out}, \mathtt{list\text{-}append}(\alpha, \beta));$
return *false*

$$\frac{E_1 \xrightarrow{\texttt{out}(\alpha)} I_1 \quad E_2 \xrightarrow{\texttt{out}(\beta)} I_2}{leq(E_1, E_2) \xrightarrow{\texttt{out}(\alpha +\!\!+ \beta)} true} (\texttt{is-leq}(I_1, I_2) \triangleright true)$$

PROCEDURE FOR leq$^{\rightarrow}$

$pm\text{-}args(E_1, E_2)$;
$single(``\rightarrow", E_1, I_1, 1)$;
$wo\_get(\texttt{out}, \alpha, 1)$;
$single(``\rightarrow", E_2, I_2, 2)$;
$wo\_get(\texttt{out}, \beta, 2)$;
$pm(\texttt{is-leq}(I_1, I_2), x_0)$;

$pm(x_0, true)$;
$wo\_set(\texttt{out}, \texttt{list-append}(\alpha, \beta))$;
return *true*

$pm(x_0, false)$;
$wo\_set(\texttt{out}, \texttt{list-append}(\alpha, \beta))$;
return *false*

$$\frac{E_1 \xrightarrow{\text{out}(\alpha)} I_1 \quad E_2 \xrightarrow{\text{out}(\beta)} I_2}{leq(E_1, E_2) \xrightarrow{\text{out}(\alpha +\!\!+ \beta)} true}\ (\texttt{is-leq}(I_1,I_2) \triangleright true)$$

PROCEDURE FOR leq$^{\rightarrow}$

$pm\text{-}args(E_1, E_2)$;
$single(\text{“}\rightarrow\text{”}, E_1, I_1, 1)$;
$wo\_get(\text{out}, \alpha, 1)$;
$single(\text{“}\rightarrow\text{”}, E_2, I_2, 2)$;
$wo\_get(\text{out}, \beta, 2)$;
$pm(\texttt{is-leq}(I_1, I_2), x_0)$;
$wo\_set(\text{out}, \texttt{list-append}(\alpha, \beta))$;

| $pm(x_0, true)$;
return *true*

| $pm(x_0, false)$;
return *false*

Efficient Interpretation of I-MSOS

via an Intermediate Language

L. Thomas van Binsbergen[1]
ltvanbinsbergen@acm.org
Neil Sculthorpe[2] Adrian Johnstone[1] Elizabeth Scott[1]

[1]Royal Holloway, University of London

[2]Nottingham Trent University

September 2, 2016