



The ECLiPSe Optimization Platform

www.eclipse-clp.org

OSSICP'08, Paris

Joachim Schimpf and Kish Shen



Outline

- Motivation
- History
- Components
 - Modelling
 - Solvers and Interfaces
- Scripting
 - Modelling
 - Search
 - Propagation
- Open Sourcing and plans



Motivation

ECLiPSe attempts to support the most common techniques used in solving Constraint (Optimization) Problems:

- CP – Constraint Programming
- MP – Mathematical Programming
- LS – Local Search
- and combinations of those

ECLiPSe is built around the
CLP (Constraint Logic Programming) paradigm



History

- 1995 ECRC (European Computer Industry Res Centre)
First ECLiPSe release 1990, predecessors earlier
Logic Programming, databases, parallelism, fd+set constraints
Shared roots with CHIP system
- 1995 – 2005 IC-Parc, Imperial College London
Focus on hybrid constraint solving, CHIC2 project
MIP integration, repair techniques, new solvers
- 1999 – 2004 Parc Technologies Ltd, London
Airline and Telecom product development
- 2004 – Cisco Systems
Networking applications
- 2006 Open-sourced www.eclipse-clp.org
Mozilla-style licence

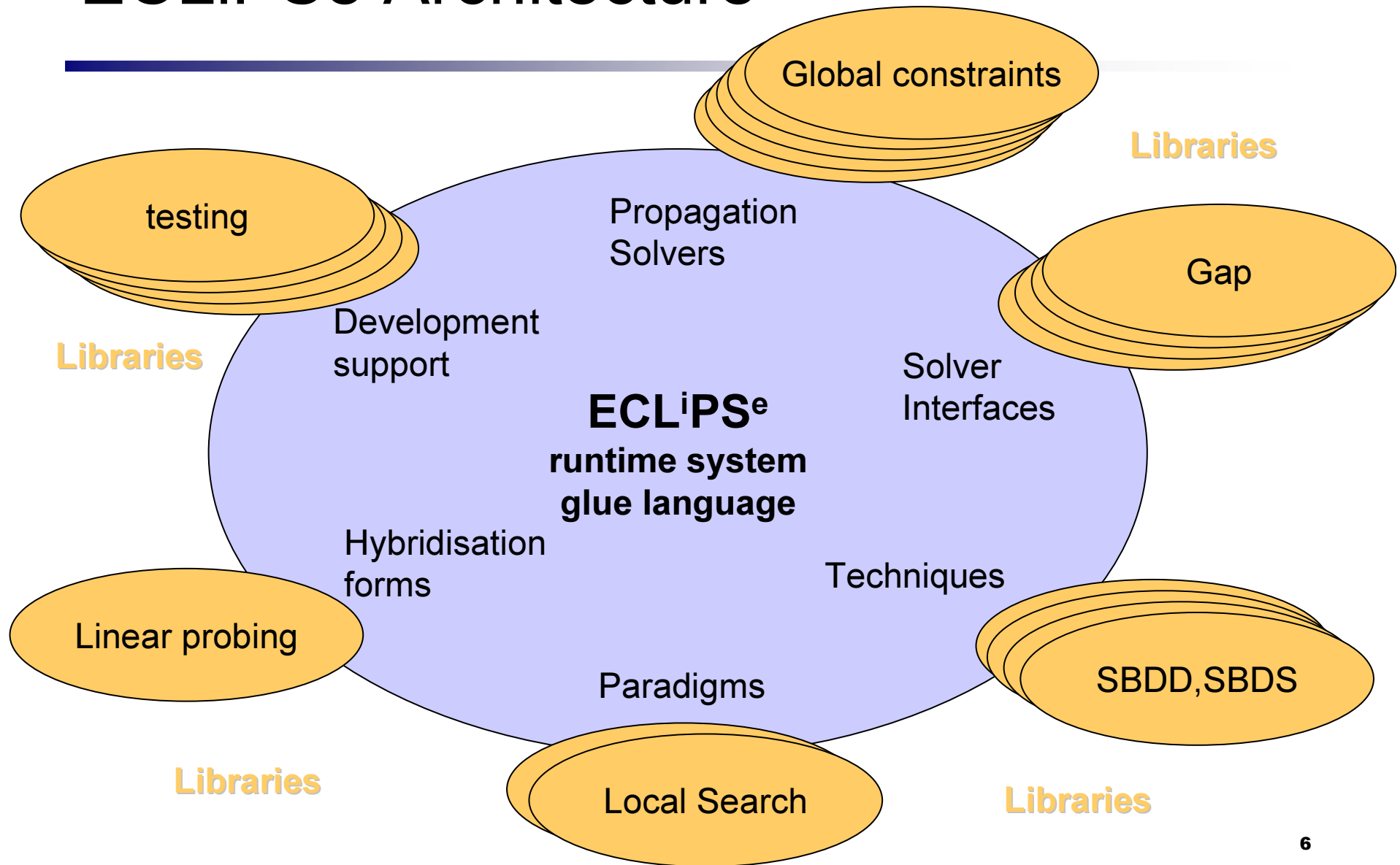


What is ECLiPSe

A **Constraint Logic Programming** system, consisting of

- A core engine
 - VM with data-driven computation, backtracking, garbage collection
- A collection of libraries
 - Solvers, algorithms, search methods, interfaces to third-party solvers
- A high-level modelling and control language
 - Logic programming based
- Development environments and tools
 - Saros (ECLiPSe/eclipse), TkECLiPSe
- Interfaces for embedding into host environments
 - Java , C/C++, Tcl/Tk

ECLiPSe Architecture

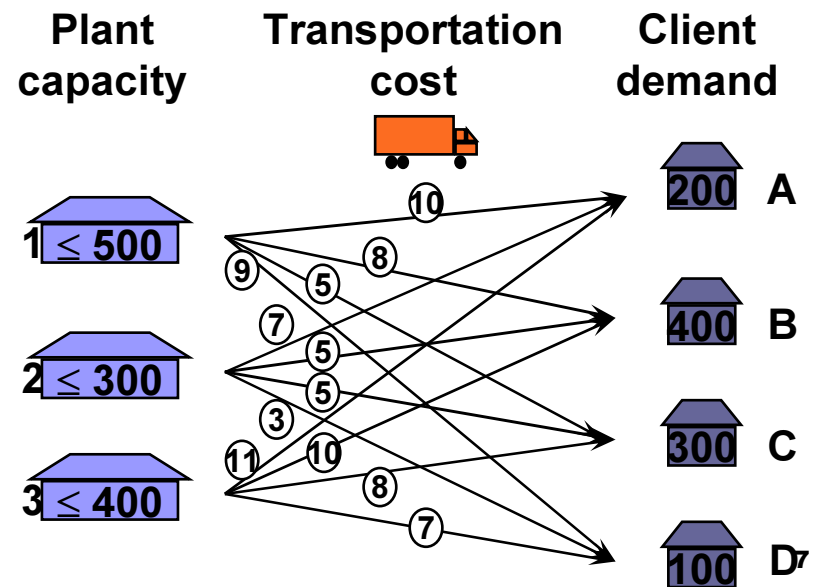


ECLiPSe as black box solver:

Flat Model, including instance data

```

model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
        8*B1 + 5*B2 + 10*B3 +
        5*C1 + 5*C2 + 8*C3 +
        9*D1 + 3*D2 + 7*D3.
    
```



ECLiPSe as black box solver: *Applying Finite Domain solver*

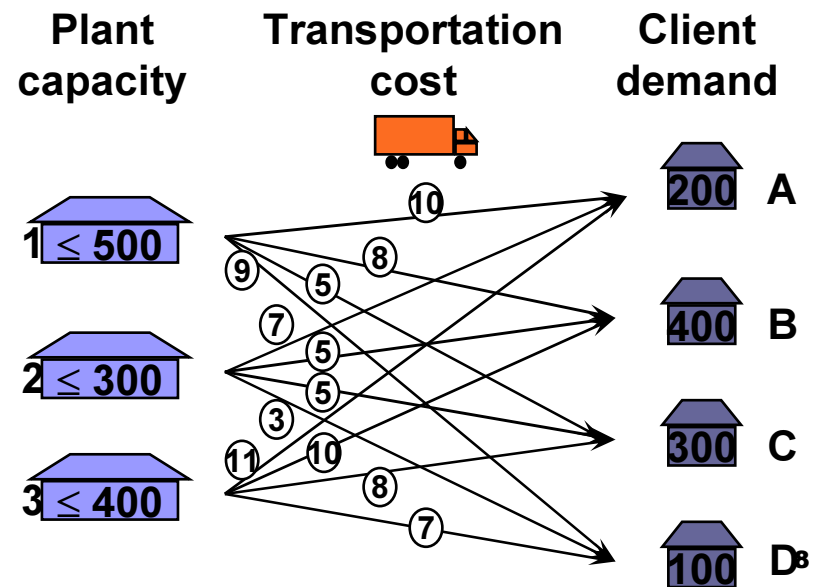
```
:- lib(ic).
:- lib(branch_and_bound).
```

Specify libraries

```
solve(Vars, Cost) :-
    model(Vars, Obj),
    Cost #= eval(Obj),
    minimize(search(Vars, 0, first_fail, indomain_split, complete, []), Cost).
```

Specify heuristics

```
model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
        8*B1 + 5*B2 + 10*B3 +
        5*C1 + 5*C2 + 8*C3 +
        9*D1 + 3*D2 + 7*D3.
```



ECLiPSe as black box solver:

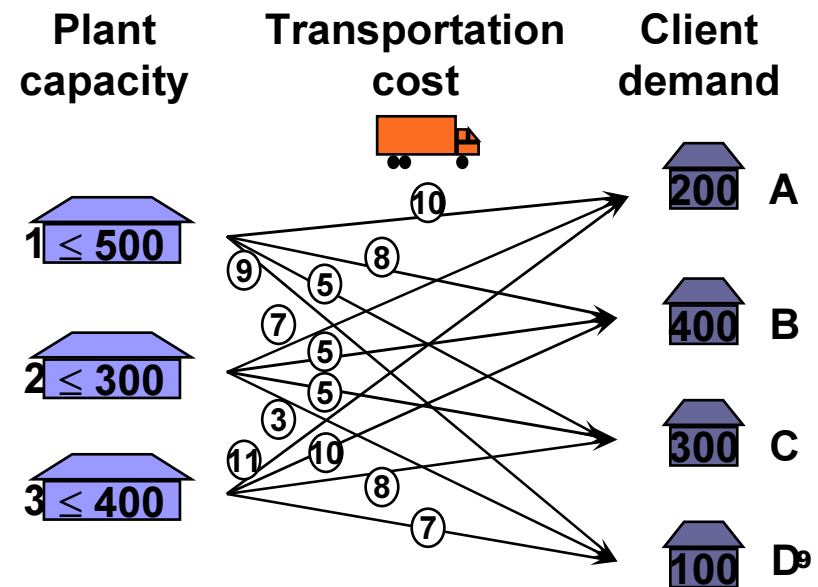
Applying LP solver

```
:- lib(eplex).
```

Specify solver library

```
solve(Vars, Cost) :-
    model(Vars, Obj),
    eplex_solver_setup(min(Obj)),
    eplex_solve(Cost).
```

```
model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
        8*B1 + 5*B2 + 10*B3 +
        5*C1 + 5*C2 + 8*C3 +
        9*D1 + 3*D2 + 7*D3.
```





ECLiPSe Language - Modelling

- Logic Programming based
 - Predicates over Logical Variables
 - Disjunction via backtracking
 - Metaprogramming (e.g. constraints as data)

$X \#> Y, \text{integers}([X,Y])$
 $X=1 ; X=2$
Constraint = $(X+Y)$
- Modelling extensions
 - Arrays
 - Structures
 - Iteration/Quantification

$M[I,J]$
task{start:S}
(foreach(X,Xs) do ...)
- Solver annotations
 - Solver libraries
 - Solver qualification

$:- \text{lib}(ic).$
[Solvers] : Constraint

One language for modelling, search, and solver implementation!



Sample Model

Generic Model, no data

```
model(Capacities, Demands, Dists, Vars, Obj) :-  
  
    length(Capacities, NP),  
    length(Demands, NC),  
    matrix(NP, NC, Plants, Clients, Vars),  
    Vars :: 0.0..inf,  
  
    ( foreach(Client, Clients), foreach(Demand, Demands) do  
        sum(Client) $= Demand  
    ),  
    ( foreach(Plant, Plants), foreach(Capacity, Capacities) do  
        sum(Plant) $=< Capacity  
    ),  
    Obj = (Dists*Vars).
```

Constraint Solver Libraries

Solver Lib	Var Domains	Constraints class	Behaviour
suspend	numeric	Arbitrary arithmetic in/dis/equalities	Passive test
fd	integer, symbol	Linear in/dis/equalities and some others	Domain propagation
ic	real, integer	Arbitrary arithmetic in/dis/equalities	Bounds/domain propagation
ic_global	integer	N-ary constraints over lists of integers	Bounds/domain propagation
ic_sets	set of integer	Set operations (subset, cardinality, union, ...)	Set-bounds propagation
grasper	graphs	Graph relations (path, scc, ...)	Propagation
ic_symbolic	ordered symbols	Dis/equality, ordering, element, ...	Bounds/domain propagation
sd	unordered symbols	Dis/equality, alldifferent	Domain propagation
propia	Inherited	any	various
eplex	real, integer	Linear in/equalities	Global, optimising
tentative	open	open	Violation monitoring



Solvers

The real/integer domain solver lib(ic)

A very general numeric solver, based on interval arithmetic

- Real-valued variables
- Integrality is a constraint
- Infinite domains supported
- Subsumes finite domain functionality

Not as fast as specialised FD solver, but general and good for hybrids.

Mathematical Programming Interface “eplex”

Features supported by solvers

	CPLEX	XPRESS	CLP/CBC	SYMPHONY/CLP
problem types	LP,MIP,QP,MIQP	LP,MIP,QP,MIQP	LP, MIP, QP*	LP, MIP
solving methods	simplex, barrier, network simplex, sifting	simplex, barrier	simplex, barrier*	simplex
Incremental changes	➤	➤	costly	costly
Probe	➤	➤	➤	➤
Col Gen Support	➤	➤	➤	➤
User defined global cuts	➤	➤	➤	➤
Infeasibility information (IIS)	all problem types	linear problems	no	no

* interfaced via solver-specific code



ECLiPSe as Glue and Scripting Language

Real-life applications involve many integration tasks

- integrating models in various forms
- integrating data and delivering results

And a lot of experimentation-heavy tasks

- specifying search strategies
- specifying heuristics
- specifying solver interaction
- specifying propagation behaviour

How to tackle these tasks

- Do-It-Yourself in your favourite language
- use a language with features suited to the task



ECLiPSe Language for Scripting

- **Logical variables** (problem variables)
with multiple attributes (domain, LP solution, reduced costs, tentative values, violations, ...)
- **Symbolic manipulation** (“meta-programming”)
to build symbolic constraints, post, decompose, inspect, etc
- **Safe arithmetic**
unlimited precision integers and rationals, safe floating point interval arithmetic
- **Search programming**
on top of built-in undo (backtrack) and copying facilities
- **Data-driven computation**
for constraint propagation, external solver triggering, visualisation
- **High-level building blocks**
solver-independent branch-and-bound, generalised propagation, specific hybridisation forms

Embedding *MiniZinc* Models

- Embedding MiniZinc strings:

```
queens(N) :-
  mzn_run_string("
    int: n;
    array [1..n] of var 1..n: q;
    constraint forall (i in 1..n, j in i+1..n)
      ( q[i] != q[j] /\ q[i]+i != q[j]+j /\ q[i]-i != q[j]-j );
    solve satisfy;
  ",
  [n = N],           % parameter map: ZincId=EclipseValue
  fzn_ic).           % solver mapping to use
```

- Model files, with parameter/variable mapping:

```
queens(N, Q) :-
  mzn_load("n_queens.mzn", fzn_ic, [n = N], [q = Q], _State),
  labeling(Q).
```

- Work with symbolic ECLiPSe-term representation of MiniZinc

```
[int:n, array([1..n] of var(1..n)):q, constraint(forall(...), (...))]
```



Search Scripting

E.g. Limited Discrepancy Search

User-defined LDS straightforward to program:

```
lds_labeling(AllVars, MaxDisc) :-
    ( fromto(Vars, Vars, RestVars, []),
      fromto(MaxDisc, Disc, RemDisc, _)
    do
      select_variable(X, Vars, RestVars),
      once select_value(X, Value),
      (
        X = Value, RemDisc = Disc
      ;
        Disc > 0, RemDisc is Disc-1, X #\= Value, indomain(X)
      )
    ).
```

Search Scripting

Restart with seeds, heuristics, limits

Jobshop example (approximation algorithm by Baptiste/LePape/Nuijten)

```
init_memory(Memory),
bb_min((
    ( no_remembered_values(Memory) ->
        once bln_labeling(c, Resources, Tasks),      % find a first solution
        remember_values(Memory, Resources, P)
    );
    scale_down(P, PLimit, PFactor, Probability), % with decreasing probability
    member(Heuristic, Heuristics),                % try several heuristics
    repeat(N),                                     % several times
    limit_backtracks(NB),                          % spending limited effort
    install_some_remembered_values(Memory, Resources, Probability),
    bb_min(
        bln_labeling(Heuristic, Resources, Tasks),
        EndDate,
        bb_options{strategy:dichotomic}
    ),
    remember_values(Memory, Resources, Probability)
),
EndDate, Tasks, TasksSol, EndApprox,
bb_options{strategy:restart}
).
```



Propagation Scripting


Arc-consistency

Arc consistency on top of weaker consistency (e.g. test, forward checking)

```
ac_constr(Xs) :-  
    (  
        weak_constr(Xs),  
        member(X, Xs),  
        indomain(X),  
        once labeling(Xs)  
    ) infers fd.
```



“Generalised Propagation” operator



Propagation Scripting

Singleton Arc-consistency

Singleton arc consistency from arc consistency, on a subproblem:

```
sac_constr(Xs) :-  
    (  
        ac_constr(<some Xs>), ..., ac_constr(<some Xs>),  
        member(X, Xs),  
        indomain(X)  
    ) infers ic.
```

If performed on the whole problem, simpler variant: *shaving*

```
shave(Xs) :-  
    ( foreach(X,Xs) do  
        findall(X, indomain(X), Values),  
        X :: Values  
    ).
```

Shaving often effective as a preprocessing step before actual search.

E.g. sudoku solvable with ac-alldifferent and shaving – no deep search needed [Simonis].



Open Sourcing Experience

- Historically, ECLiPSe was proprietary from the start
- But research centre couldn't provide commercial support
- Parent/sponsor companies didn't want to sell tool and/or support
- Academic licences, but uptake limited by bureaucracy
- Academic licences, but no commercial upgrade path
- Small user base within parent companies, not enough testers
- But: exclusivity was nevertheless considered important by spin-out investors and academic IPR exploitation outfits
- In retrospect, community uptake and reputation of system suffered
- Possibly also negative consequences for robustness and feature set



Now and Next

- Technical

- Interface to COIN/OR solvers still being polished and extended

- MiniZinc link recently released

- ECLiPSe release 6.0 forthcoming, includes a new compiler

- Saros development environment – released, needs manpower

- New solver interfaces (MiniSat, Gecode) planned

- Organisational

- Cisco continues support and maintenance

- 2 academic collaboration projects currently in selection phase

- Wider contributions sought



ECLiPSe Resources

- Open-sourced (MPL) by Cisco 9/2006
 - Sources and binaries on www.sourceforge.net/eclipse-clp
 - Project web site www.eclipse-clp.org
- Open-source user community
 - Traditionally academic teaching/research
 - Audience widening since open-sourcing
- Book
 - Constraint Logic Programming using ECLiPSe
 - Krzysztof Apt & Mark Wallace, Cambridge University Press, 2006.