Reading Paths and Teaching

Extra Material and Dependencies

You will find at the end of this book a list of notations and a summary of ASP syntax and semantics that should provide a convenient quick reference (Index of Notations, Syntax, Operational Semantics). This is followed by a graphical view of ASP properties (page 331), and the syntax of ASP extensions (Synchronizations, Migration, Groups, Components).

The Appendices detail formal definitions and proofs of the main theorems and properties introduced in Part III.

Figure 1 exhibits the dependencies between chapters and sections. Each chapter is best read after the preceding chapters. For example, in order to fully understand the group communication in ASP (Chap. 13), one should read Chaps 3, 4, Part III (Chaps. 6, 7, 8, 9), and Chap.10. Going down the lines (Fig. 1), one can follow the outcomes of chapters. For instance, still for group communication in Chap. 13, immediate benefits are parallel components (Sect. 14.5), and a practical implementation of typed group communication within ProActive (Chap. 16).

Text Book

Besides researchers and middleware designers, the material here can also be used as a text book for courses related to *models, calculi, languages for concurrency, parallelism, and distribution.* The focus is clearly on recent advances, especially object-orientation and asynchronous communications. Such courses can provide theoretical foundations, together with a perspective on practical programming and software engineering issues, such as distributed components.

The courses cover classical calculi such as CSP [88] and π-calculus [119, 120, 144], object-orientation using ς-calculus [3, 1, 2], and ASP [52], and advanced issues such as mobility, groups, and components. Overall, the objectives are threefold:

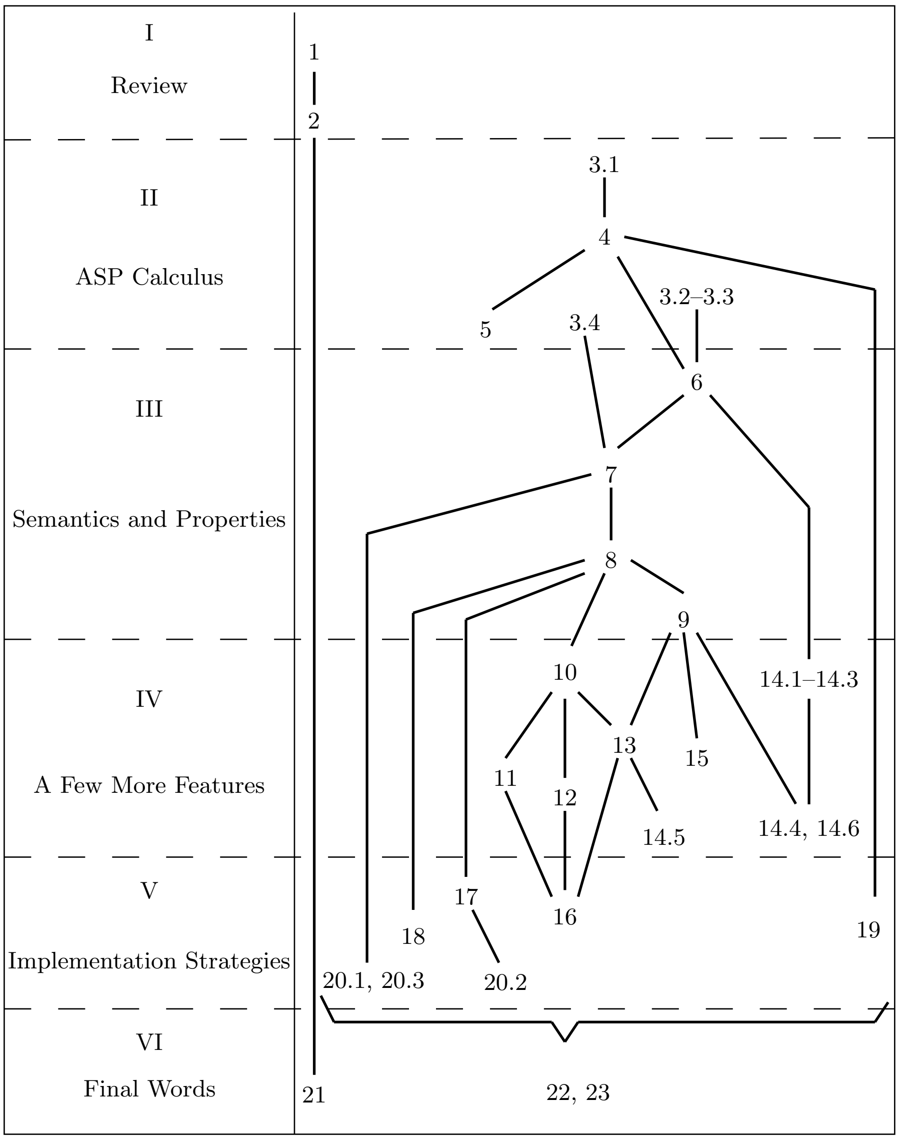

Fig. 1. Suggested reading paths

(1) study and analyze existing models of concurrency and distribution,
(2) survey their formal definitions within a few calculi,
(3) understand the implications on programming issues.

Depending on the objectives, the courses can be aimed at more theoretical aspects, up to proofs of convergence and determinacy within π-calculus and ASP, or targeted at more pragmatic grounds, up to practical programming sessions using software such as PICT [132, 131] or *ProActive* [134].

Below is a suggested outline for a semester course, with references to online material, and chapters or sections of this book:

Models, Calculi, Languages for
Concurrency, Parallelism, and Distribution

1.	Introduction to Distribution, Parallelism, Concurrency General Overview of Basic formalisms	1 [39]
2.	CCS, and/or Pi-Calculus	2.1.3 [73]
3.	Other Concurrent Calculi and Languages (Process Network, Multilist, Ambient, Join, ...)	2.1.4, 2.2 [125]
4.	Object-Oriented calculus: ς-calculus	2.1.5 [4]
5.	Overview of Concurrent Object Calculi (Actors, ABCL, Obliq and Øjeblik, $\pi o \beta \lambda$, **conc**ς-calculus, ...)	2.1.2, 2.3 [39]
6.	Asynchronous Method Calls and Wait-by-necessity ASP: Asynchronous Sequential Processes	3, 4, 5
7.	Semantics, Confluence, Determinacy	6, 7, 8, 9
8.	Advanced issues I: Confluent and non-confluent features, mobility	10, 11, 12 [125]
9.	Advanced issues II: Groups, Components	13, 14
10.	Open issue: reconfiguration Conclusion, Perspective, Wrap-up	15, 21, 22, 23

The Web page [39] gathers a broad range of information aimed at concurrent systems, also featuring parallel and distributed aspects. Valuable material for teaching models of concurrent computation, including CCS and π-calculus can be found at [73]. The Web page [4] is dedicated to the book *A Theory of Objects* [3]; it references pointers to courses using ς-calculus, some with teaching material available online. Finally, a comprehensive set of resources related to calculi for mobile processes is available at [125].

Assignments can include proofs of the confluence or non-confluence natures of a few features (e.g., delegation, explicit wait, method update, testing future or request reception, non-blocking services, join constructs, etc.). More practical assignments can involve designing and evaluating new future-update strategies, new request delivery protocols, or new schemes for pipelining control. Practicality can reach as far as implementing examples or prototypes, using PICT [132, 131], *ProActive* [134], or other programming frameworks.

A Theory of Distributed Objects online

We intend to maintain a Web page for general information, typos, etc. Extra material is also expected to be added (slides, exercises and assignments, contributions, reference to new related papers, etc.). This page is located at:
`http://www.inria.fr/oasis/caromel/TDO`

Do not hesitate to contact us to comment or to exchange information!