Answer-set programming: themes and challenges

Mirosław Truszczyński

University of Kentucky

NMR-04

Whistler, Canada, June 6, 2004

ASP phenomenon

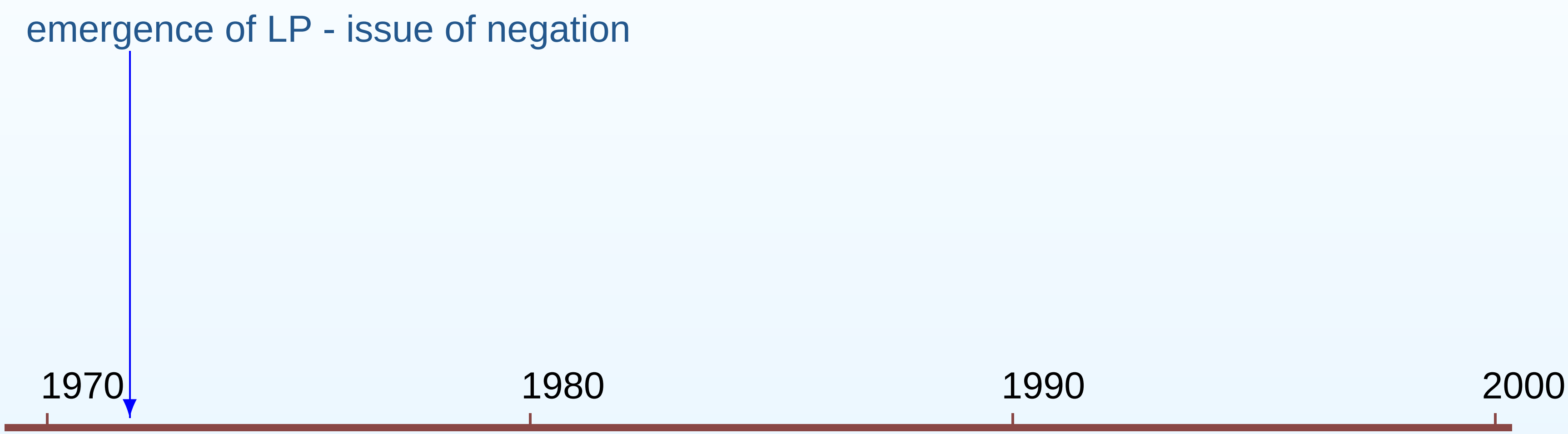
- ASP emerged around 1999 and quickly became a thriving research area
  - resuscitated logic-based NMR
  - new results, many papers, new people, growing recognition
- What is it exactly and what happened?

ASP paradigm

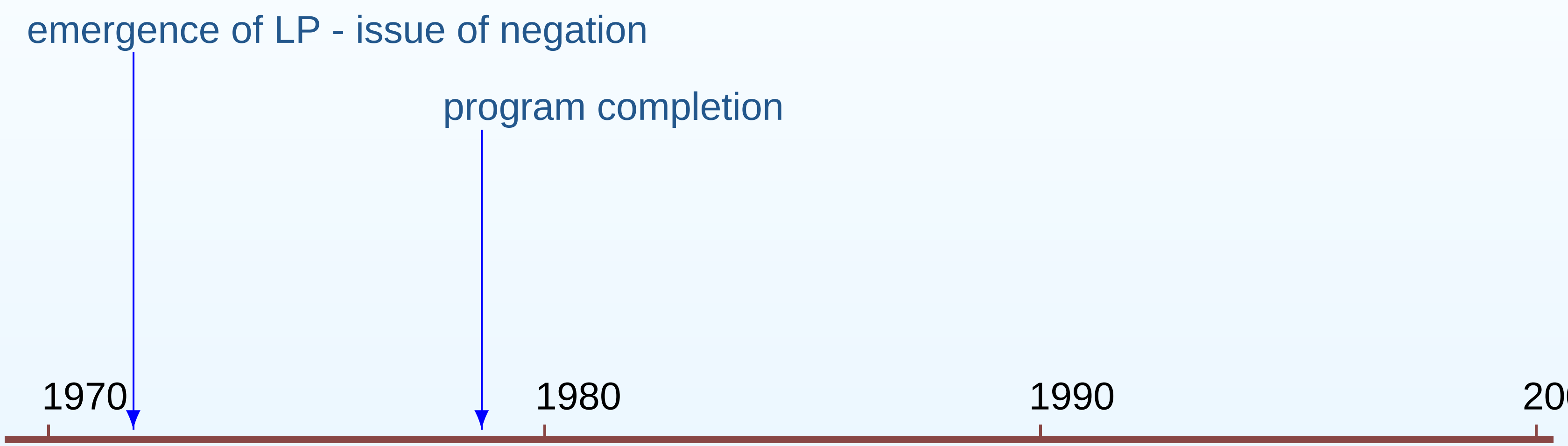
- ASP — a declarative computational approach to knowledge representation
- More broadly — declarative programming approach for solving search problems
- Defining features:
  - high-level modeling language
  - distinct interpretation: *theories encode search problems so that models represent solutions*
  - uniform control: computing models

A brief history of ASP

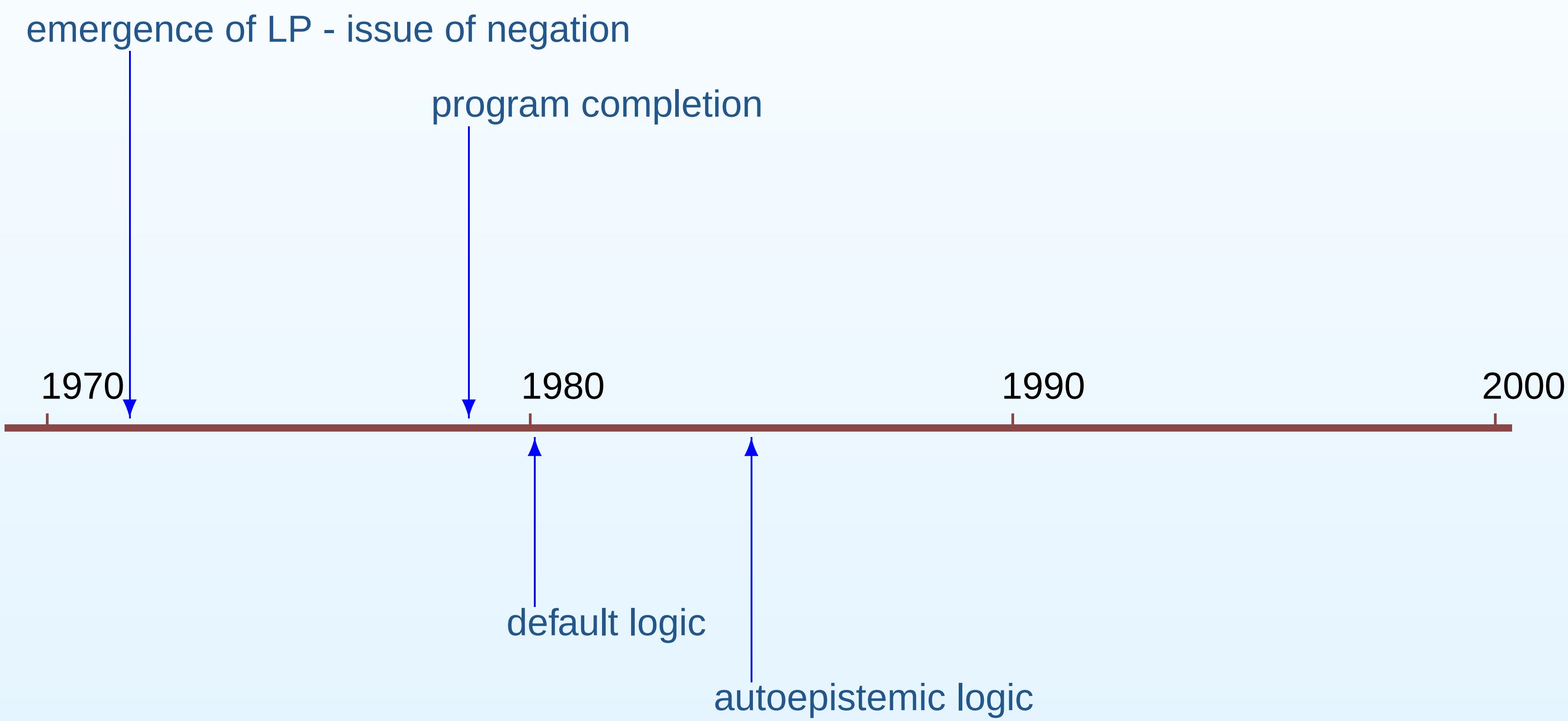
A brief history of ASP



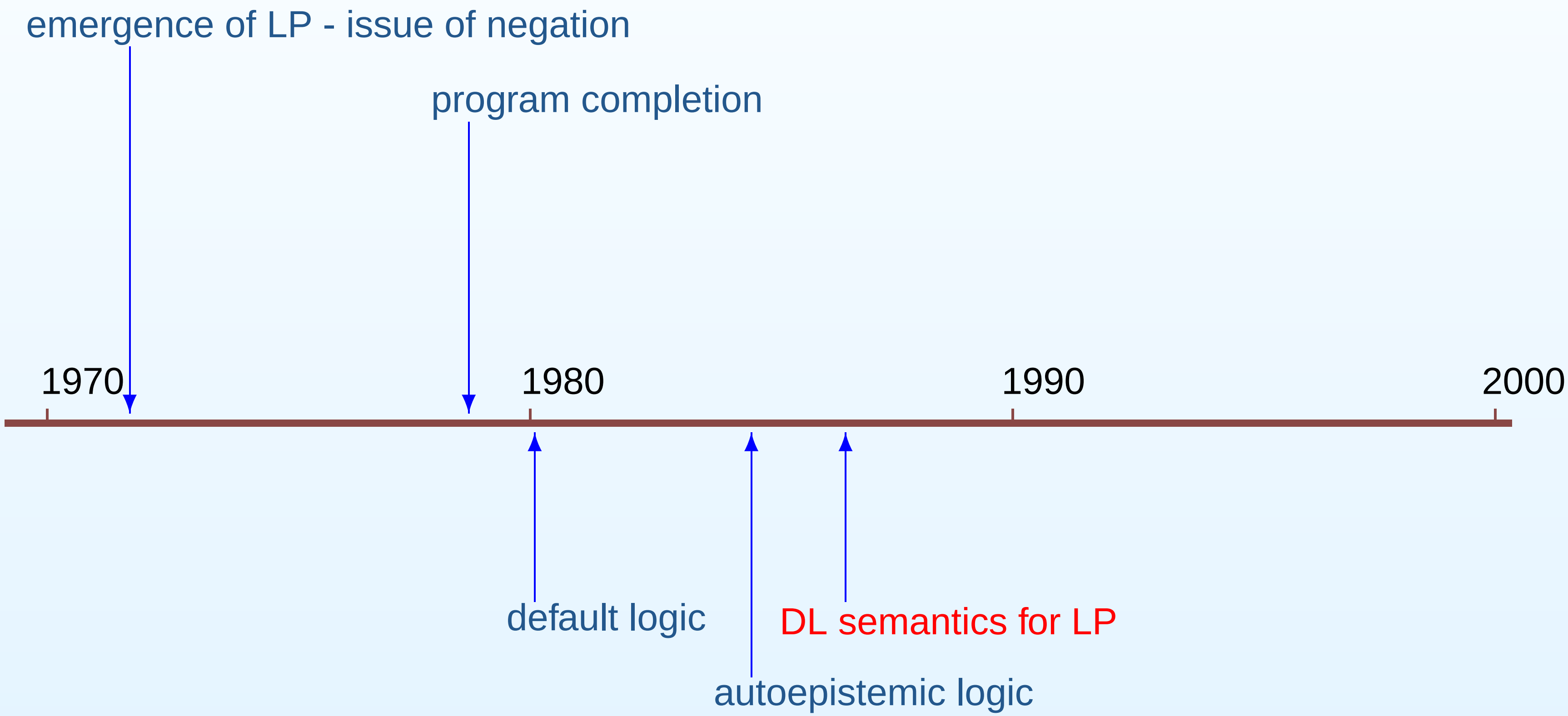
A brief history of ASP



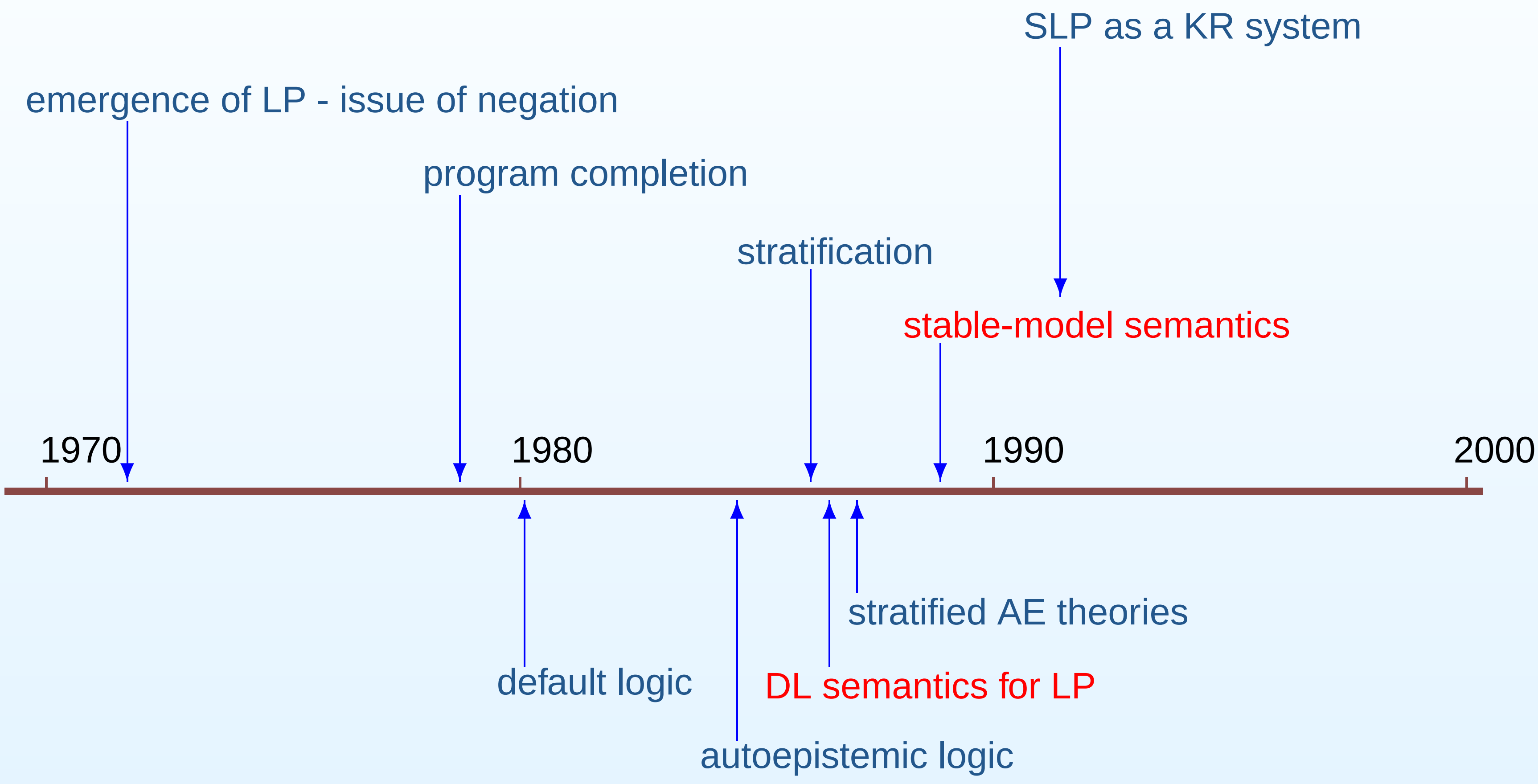
A brief history of ASP



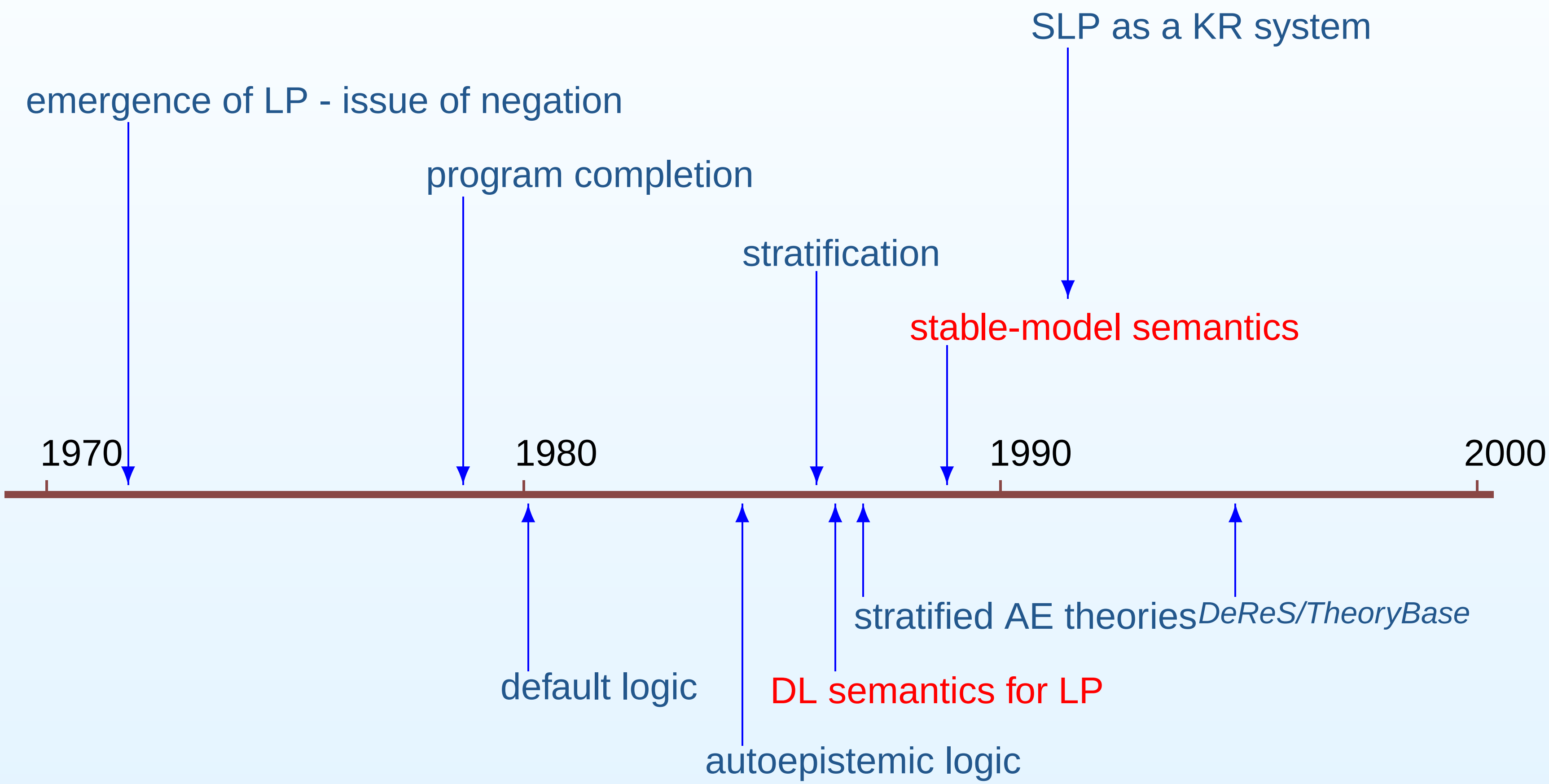
A brief history of ASP



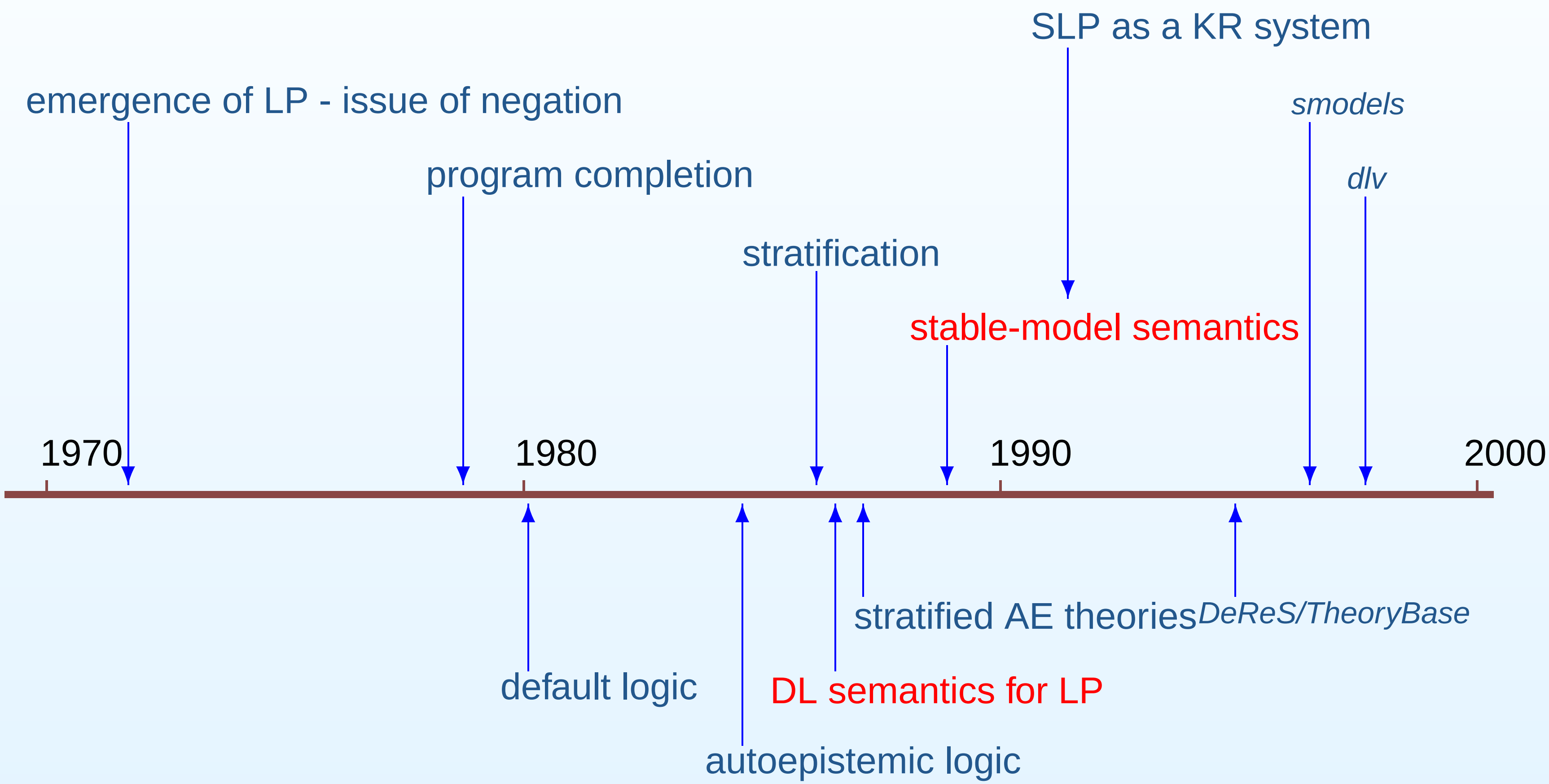
A brief history of ASP



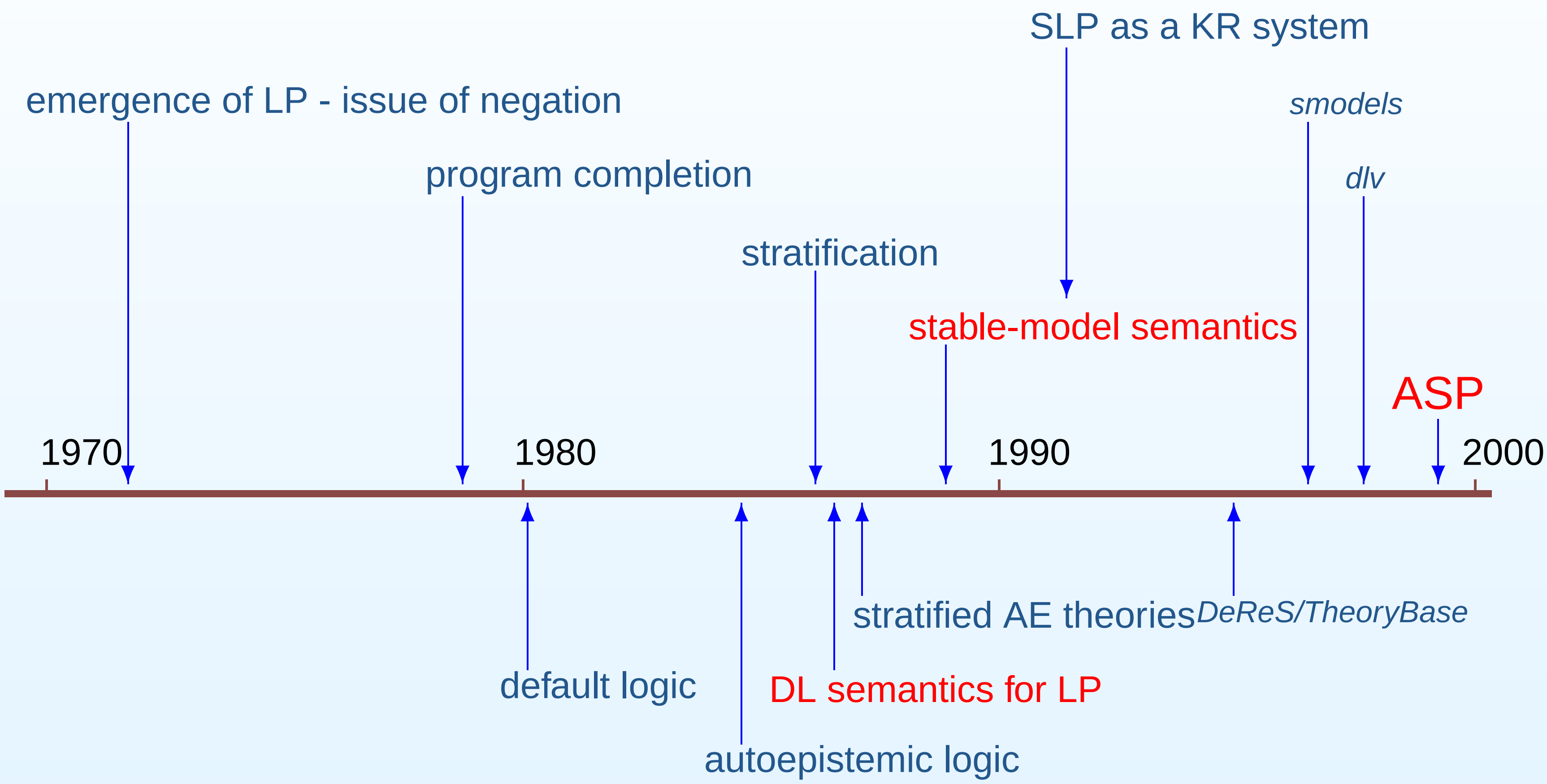
A brief history of ASP



A brief history of ASP



A brief history of ASP



ASP —five years later

- Exciting theoretical results
- New algorithms
- Aggregates
- New formalisms — beyond logic programming
- Emerging connections to SAT and CSP
- Successful applications

Program equivalence

- How to rewrite programs?
- How to optimize programs?
- Towards programming methodology
- Program equivalence, strong equivalence, uniform equivalence
  (Lifschitz, Pearce, Valverde; Lin; Turner; Osorio, Navarro, Arrazola; Eiter, Fink, Tompits, Woltran)

Program equivalence

- Disjunctive programs P and Q are *equivalent* if P and Q have the same answer sets
- Fundamental question: how to simplify (rewrite) logic programs preserving equivalence
- Programs P and Q are *strongly equivalent* if for every program R, answer sets of $P \cup R$ coincide with answer sets of $Q \cup R$
  - Replacing a subprogram with a strongly equivalent one preserves equivalence
- Disjunctive programs P and Q are *uniformly equivalent* if for every set of atoms X, answer sets of $P \cup X$ coincide with answer sets of $Q \cup X$
  - Replacing the set of rules of the program (intentional part) with a uniformly equivalent one preserves equivalence

Strong equivalence

- A pair of sets of atoms (X, Y) is an *SE-model* of a DLP P if
  - $X \subseteq Y$
  - $Y \models P$
  - $X \models P^Y$
- Two DLPs P and Q are strongly equivalent if and only if $SE(P) = SE(Q)$
- Connections to the logic “here-and-there” and to the logic S4F

Uniform equivalence

- An SE-model (X, Y) of a DLP P is a *UE-model* of P if for every $(X', Y) \in SE(P)$, where $X \subseteq X' \subseteq Y$, $X' = X$ or $X' = Y$
- Two *finite* DLPs P and Q are uniformly equivalent if and only if they have the same UE models
- The general case is also resolved

Additional comments

- Most of program transformations preserve strong and uniform equivalence (TAUT, RED$^-$, NONMIN, CONTRA, WGPPE); some do not (RED$^+$, GPPE)
  Osorio, Navarro, Arrazola; Eiter, Fink, Tompits, Woltran)
- Further generalizations possible (Turner - lparse programs)
- Complexity is well understood (Turner; Lin; Eiter, Fink)
  - given two NLPs, deciding whether they are strongly equivalent is coNP-complete (holds, in fact, for DLPs)
  - given two DLPs, deciding whether they are uniformly equivalent is Π_2^P-complete
  - given two DLPs that are head-cycle free, deciding whether they are uniformly equivalent is coNP-complete
- ASP can be used to test equivalence! (Janhunen, Oikarinen)

SLP and propositional logic

- Let Π_n consist of:

$$p_{ijk} \leftarrow \mathbf{not}(q_{ijk})$$
$$q_{ijk} \leftarrow \mathbf{not}(p_{ijk})$$
$$r_1$$
$$r_k \leftarrow r_i, r_j, p_{ijk}$$

- If $P \not\subseteq NC^1/poly$ (that is, not all languages in P can be recognized by polynomial size propositional formulas)
- Then it is impossible to find a sequence of propositional formulas $F_1, F_2, \ldots$ such that
  - for every n, the satisfying assignments for F_n are identical to the answer sets for Π_n
  - the sizes of the formulas F_n are bounded by a polynomial in n

  (Lifschitz and Razborov)
- Related to earlier work on compilability and succinctness

Semantic foundations

- Universal algebra of lattices, operators, approximation operators and fixpoints (Denecker, Marek, MT; influenced by Fitting's work on LP)
- Uniform abstract approach to nonmonotonic reasoning systems
- Full understanding of the relationship between DL and AEL

$$
\begin{tikzcd}
& \mathcal{E}_\Delta \arrow[dl] \arrow[dr] & & & \mathcal{D}_T \arrow[dl] \arrow[dr] & \\
E_\Delta & & \mathcal{C}_\Delta \arrow[dl, leftrightarrow] & D_T & & \mathcal{C}_T \arrow[dl, leftrightarrow] \\
& C_\Delta & & & C_T &
\end{tikzcd}
$$

$$\longrightarrow$$

$$\frac{\alpha : \beta}{\gamma} \Rightarrow K\alpha \wedge \neg K \neg \beta \supset \gamma$$

- Ultimate well-founded semantics and ultimate stable-model semantics
- Generalizations to handle programs with constraints?
- Formalization of the notion of non-monotone induction (Denecker)

Computing

- Native solvers
  - *smodels* (Niemelä, Simons, Syrjänen, Soininen)
  - *dlv* (Eiter, Leone, Mateis, Pfeifer, Scarcello, Faber, Dell'Armi, Ielpa)
  - *NoMoRe* (Linke, Schaub, Anger, Konczak, Bösel)
  - adapting advances in SAT — learning (Schlipf, Ward)
- Direct use of SAT solvers
  - compiling LPs into SAT (Ben-Eliyahu; Janhunen)
  - bringing together program completion, Fages Lemma, loop formulas and SAT (Lifschitz, McCain, Turner, Erdem, Lierler, Lee; Lin, Zhao; Lierler, Maratea, Giunchiglia)

SAT —take one

- Exploit concepts of program completion and tightness
- For tight logic programs supported and stable models coincide (Fages)
- Supported models of a logic program are models of this program completion
- Thus, computing stable models of a tight logic program can be accomplished by computing models of the completion
  - *cmodels* (earlier used in *ccalc*)
  - Some additional propositional variables may be necessary when converting the completion formula into a CNF (typically, not a big problem)
  - May fail for non-tight programs (a slightly more general version of the approach possible but it still does not cover all cases)

SAT —take two: loop formulas

- Dependency graph for a program P — $G(P)$
  - atoms are vertices
  - arc from p to q if there is a rule with the head p and with q in the positive body
- Loop — any strongly connected subgraph of $G(P)$
- Loop formula for a loop L
  - $R^-(L)$ — all rules about atoms in L whose edges point outside L
  - B_p — disjunction of bodies of all rules in $R^-(L)$ that define p
  - $\Phi_L = \bigvee_{p \in L} p \supset \bigvee_p B_p$
  - Informally, if at least one atom L is in a stable model, there must be an atom p in L such that at least one rule defining p must have all atoms of its positive body outside of L (is in $R^-(L)$)
- Loop theorem: M is a stable model of P if and only if it is a model of $Comp(P) \cup \{\Phi_L : L \in L(P)\}$

How to implement it?

- There may be exponentially many loops
- But one can proceed incrementally!
    1. $T := \mathit{comp}(P)$
    2. Find model M of T; terminate with failure, otherwise
    3. If M is an answer set, output M; terminate
    4. Otherwise, compute a loop L such that $M \not\models \Phi_L$
    5. $T := T \cup \{\Phi_L\}$; go back to step 2.
- Loops needed in (4) can be computed quickly
- In the worst case, exponentially many steps needed
- Typically, if stable models exist — much better performance
- If not — a potential problem

A way around the problem

- Do not use loop formulas at all
  - Apply a DPLL procedure for $\mathit{comp}(P)$
  - Test each computed model M for stability
  - Continue accordingly (continue search or output the model and stop)
- Can be improved if DPLL with learning is used
  - each time M is not a stable model, learn a conflict clause
  - a conflict clause can be computed with the help of loop formulas
  - implement a scheme to forget (delete) some conflict clauses as the search goes on

The idea extends!!

- Disjunctive logic programming
  - completion
  - dependency graph, loop
  - loop formula
- Circumscription

What’s behind the success of *smodels*?

- Performance of *smodels* (including *lparse*)
- Modeling capabilities
- Both aspects strongly depend on the use of cardinality and weight constraints
- Which brings us to the next theme ... aggregates (Niemelä, Soininen, Simons; Pelov, Denecker, Bruynooghe; Dell’Armi, Faber, Ielpa, Leone, Pfeifer)

Abstract constraints

- At — a fixed set of propositional atoms
- *Abstract constraint* — a collection of subsets of At
  - $even = \{X \subseteq At\colon |X|$ is even$\}$
  - "At least k" constraint: $\{X\colon X \subseteq At;\ k \leq |X|\}$
- An *abstract constraint atom* — an expression $C(X)$, where
  - C is an abstract constraint
  - X is a finite subset of At — the *scope* of $C(X)$
- A *rule with abstract constraint atoms*:

$$H \leftarrow A_1, \ldots, A_m, \mathbf{not}(B_1), \ldots, \mathbf{not}(B_n)$$

When it makes sense and what we get

- Under restriction to monotone and consistent atoms
  $C(X)$ is monotone if C is closed under superset
  $C(X)$ is consistent if for some $Y \subseteq X$, $Y \in C$
  we get direct generalization of normal logic programs (uniform with respect to models, supported models and stable models)
- Under simple transformations — generalization of logic programs with weight constraints
  - basis for the theory of such programs
- The theory is developed in terms of *non-deterministic* operators on the lattice of interpretations
- Can be further generalized to the language of nondeterministic operators on complete lattices and their fixpoints
- Does the approximation theory generalize?

Languages for ASP —beyond logic programming

- Predicate logic extended with (limited) CWA — *aspps* (East, MT)
- Logic ESO — existential fragment of second order logic (Cadoli, Mancini, Schaerf)

aspps system

- Program

  $pred\ invc(vtx).$
  $var\ X.$

  $\{invc(X)[X] \colon vtx(X)\}k.$
  $edge(X,Y) \rightarrow invc(X) \vee invc(Y).$
- Grounding — *psgrnd*
- Solving — *aspps*
- Easy to use off-the-shelf SAT and PB-SAT solvers
- Effective local-search methods — *wsat(cc)*
- The same expressive power as that of SLP (class NPMV)
- But, can predicate logic approaches be competitive on KR applications?
  - negation-as-failure?
  - transitive closure

Applications

- Knowledge representation
  - reasoning about action, planning and diagnosis — ASP particularly appropriate (Giunchiglia, Lee, Lifschitz, McCain, Turner; Baral; Gelfond; Faber, Leone, Pfeifer, Polleres)
  - qualitative decision theory — elicitation of and reasoning about preferences (Brewka; Eiter, Brewka; Delgrande, Schaub, Tompits; Gelfond, Son; Inoue, Sakama; Brewka, Niemelä, MT)
    - representing preferences, specifying orders on answer sets
    - ASP as a uniform computational tool
    - relation to CP-network approach
- Product configuration (Soininen, Sulonen, Tiihonen, Niemelä)
  - *smodels* as a computational engine
  - *Variantum* — a recent spin-off

Applications

- Bounded model checking
  - linear-time logic compiled into a linear-size logic program
    Heljanko, Niemelä
  - built-in transitive closure is crucial!
- Combinatorics — computing van der Waerden numbers (Dransfi eld, Marek, Liu, MT)
  - $W(2,6) \geq 342$

Challenges

Random logic programs

- Propose models of random logic programs with constraints
  - must lead to a “hard” region
- Possibly already solved in the case of normal logic programs (Lin and Zhao)
  - k-LP(n, m) — rules of length k, n atoms, m rules
  - randomly select an atom for the head
  - randomly select $k - 1$ *different* atoms for the body
  - negate each with probability 0.5
  - if the rule is new — include it
  - repeat to get m rules
- Establish bounds on the location of the hard region

Program-rewriting techniques

- Develop principles under which replacing programs with strongly (uniformly) equivalent ones leads to programs with better computational properties
- Develop program-rewriting techniques at the predicate level

Non-deterministic operators on lattices

- Establish a formal theory of non-deterministic operators on lattices; generalize approximation theory to that setting
  (towards an abstract treatment of programs with aggregates)

Importance of transitive closure

- What is really behind the effectiveness of LP-based ASP?
- Is it default negation or transitive closure? Or both?
- My guess: it is transitive closure!

Algorithms

- Design native local-search methods to compute stable models (seems difficult; work by Dimopoulos and Sideris not conclusive)
- Develop new generation of complete algorithms for computing stable models with aggregates
  - better implementation of unit propagation (wfs in liear time?)
  - stronger propagation methods (ultimate wfs?)
  - dynamic backtracking, backjumping
  - branching heuristics (which heuristics, when they work and why)
  - conflict-clause learning
- Exploit program structure to enhance processing
  - one of features of ASP that SAT does not have

Computational benchmarks

- $S(5)$ and $W(5,3)$
  - $S(5) \geq 160$; $W(5,3) \geq 125$
  - are they equalities?
- Wire-routing on 50×50 grids with obstacles and with 30 terminal pairs
- 15-puzzle problem with plans of length 40 and more
- Random logic programs with 500 atoms selected from the hard region
- All SAT benchmarks

Programming support

- Build programming interfaces
  - support for modeling, debugging and optimizing programs
  - integration with other programming environments

Community

- Bringing togather SAT and ASP
  - SAT
    - fine-tuned data structures (watched literals)
    - learning
    - local-search methods, ...
  - ASP
    - modeling languages
    - default negation, transitive closure
    - stronger propagation techniques
  - More cross-fertilization needed
- ASPARAGUS — towards objective experimentation and benchmarking

Thank you!