SMART Systems (Vorlesung: KI & XPS)

Ralf Möller, Univ. of Applied Sciences, FH-Wedel

- Beim vorigen Mal:
  - Grundlagen von XML-Schema und DAML
- Inhalt heute:
  - Semantic Web und DAML (Teil 1)
  - Algorithmus für ALCN-Konsistenztest
  - Suchstrategien für Konsistenztest-Implementierung
- Lernziele:
  - Grundlagen von zukünftigen Informationsverarbeitungsprinzipien

Acknowledgments

- Diese Vorlesung enthält Material von
  - I. Horrocks (Univ. Manchester) und
    U. Sattler (RWTH Aachen)

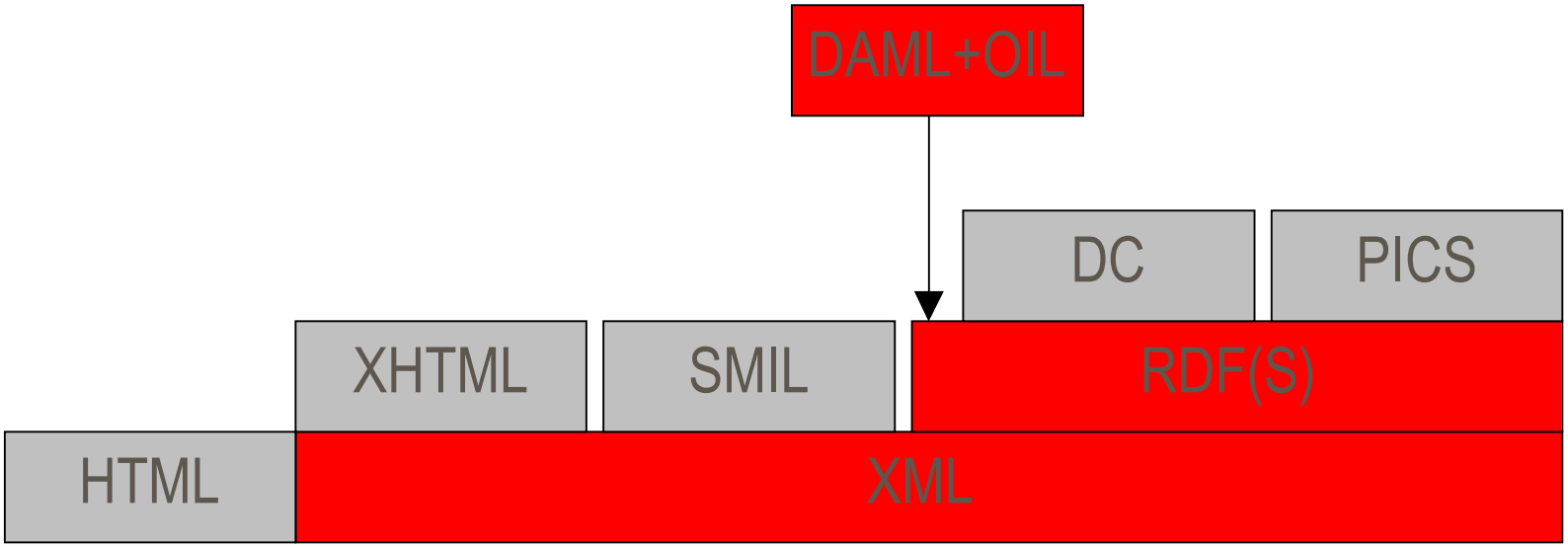
DAML+OIL Design Objectives

- Well ***designed***
  - Intuitive to (human) users
  - Adequate expressive power
  - Support machine understanding/reasoning
- Well ***defined***
  - Clearly specified syntax (obviously)
  - Formal semantics (equally important)
- ***Extend*** existing web standards
  - DAML+OIL is built on top of RDF(S)

Why Build on RDF

- Provides basic ontological primitives
  - Classes and relations (properties)
  - Class (and property) hierarchy
- Can exploit existing RDF infrastructure
- Provides mechanism for using ontologies
  - RDF triples assert facts about resources
  - Use vocabulary from DAML+OIL ontologies

The Cake!



Why RDF Is Not Enough

- Expressive inadequacy
  - Only range/domain constraints (on properties)
  - No properties of properties (unique, transitive, inverse etc.)
  - No equivalence, disjointness, coverings etc.
  - No necessary and sufficient conditions (for class membership)
- Poorly (un) defined semantics

How DAML+OIL Builds ON RDFS (1)

- Extends expressive power
  - Constraints (restrictions) on properties of classes (existential/universal/cardinality)
  - Boolean combinations of classes and restrictions
  - Equivalence, disjointness, coverings
  - Necessary and sufficient conditions
  - Constraints on properties

How DAML+OIL Builds ON RDFS (2)

- Provides well defined semantics
  - Meaning of DAML+OIL statements is formally specified
  - Both model theoretic and axiomatic specifations provided
  - Allows for machine understanding and automated reasoning

DAML+OIL ↔ RDF

- DAML+OIL ontology is a set of RDF statements
- DAML+OIL defines semantics for certain statements
- Does **NOT** restrict what can be said
  - Ontology can include arbitrary RDF
- But no semantics for non-DAML+OIL statements

Well Designed(?)

- Intuitive to (human) users
  - Supports common ontological idioms
- Adequate expressive power
  - Extends RDF in several directions
- Support for machine understanding/reasoning
  - Designed to be "implementable"
  - No features for which it is difficult or impossible to define clear semantics (e.g., defaults)
  - Decidable and (empirically) tractable reasoning

Why Automated Reasoning?

- "Semantic web" requires machine understanding (of resource descriptions)
  - Reasoning is integral to understanding
- Supports design and use of ontologies
  - Checking class consistency
  - Checking/deriving subClassOf hierarchy
  - Particularly useful when ontologies are large, multi-authored and rapidly evolving
  - Also useful when integrating/sharing ontologies
- Does not tell us how to deal with inconsistencies
  - But we should be able to determine when they exist

DAML-Zusammenfassung (Klassenkonstruktoren)

Constructor	Abbreviation	Example
intersectionOf	$C_1 \wedge \ldots \wedge C_n$	Human $\wedge$ Male
unionOf	$C_1 \vee \ldots \vee C_n$	Doctor $\vee$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
toClass	$\forall P.C$	$\forall$hasChild.Doctor
hasClass	$\exists P.C$	$\exists$hasChild.Lawyer
hasValue	$\exists P.\{x\}$	$\exists$citizenOf.{USA}
minCardinalityQ	$\geqslant n\, P.C$	$\geqslant 2$ hasChild.Lawyer
maxCardinalityQ	$\leqslant n\, P.C$	$\leqslant 1$ hasChild.Male
cardinalityQ	$=n\, P.C$	$=1$ hasParent.Female

DAML-Zusammenfassung (Axiome)

Axiom	Abbreviation	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\wedge$ Biped
sameClassAs	$C_1 \doteq C_2$	Man $\doteq$ Human $\wedge$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
samePropertyAs	$P_1 \doteq P_2$	cost $\doteq$ price
sameIndividualAs	$x_1 \doteq x_2$	President_Bush $\doteq$ G_W_Bush
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
differentIndividualFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
inverseOf	$P_1 \doteq P_2^-$	hasChild $\doteq$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
uniqueProperty	Thing $\sqsubseteq \leqslant 1 P$	Thing $\sqsubseteq \leqslant 1$hasMother
UnambiguousProperty	Thing $\sqsubseteq \leqslant 1 P^-$	Thing $\sqsubseteq \leqslant 1$isMotherOf$^-$

Extensionale Beschreibungen: One-of (1)

Motivation

Manchmal möchte man Begriffe *extensional*, durch Aufzählung ihrer Instanzen, beschreiben, z.B., die Menge der Ampelfarben ist `ROT`, `GRUEN`, `GELB`.

Idee

Eine neue Symbolmenge, die *Individuennamen* ($i, i_1, \ldots$) enthält. Diese Individuennamen werden als Elemente des Universums interpretiert (siehe für die genauen Bed. bei *Assertionen*).

Man kann dann einen Begriff durch Aufzählung von Individuen beschreiben.

Extensionale Beschreibungen: One-of (2)

Syntax

Sind $i_1, \ldots, i_n$ Individuennamen, so ist das folgende eine Begriffsbeschreibung:

- $\{i_1, \ldots, i_n\}$

Semantik

- $\{i_1, \ldots, i_n\}^{\mathcal{I}} = \{i_1^{\mathcal{I}}, \ldots, i_n^{\mathcal{I}}\}$

Anwendung

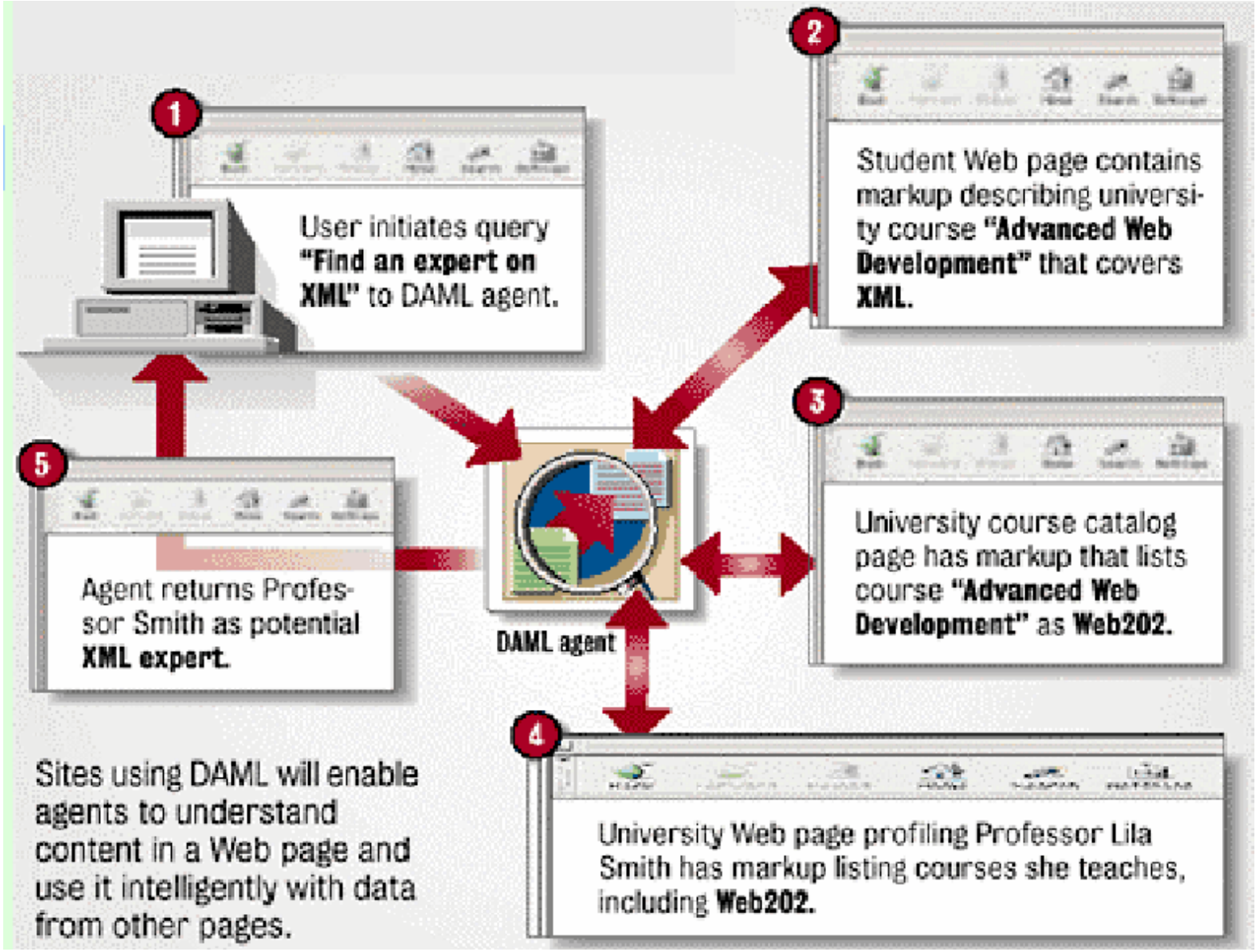
Die Ampelfarben könnten folgendermaßen definiert werden:

- $\{\texttt{ROT}, \texttt{GELB}, \texttt{GRUEN}\}$

Extending DAML+OIL

- Work in progress on Datatypes
  - Plan to support (some of) XML-Schema datatypes
  - Datatypes will be disjoint from "abstract" classes and only accessible via properties
  - Maintains "implementability" of language
- Further extensions in new language layers
  - E.g., DAML-RULES
  - Layers will use DAML+OIL as it uses RDF

Semantic Web-Anwendungsszenario



Basis: Consistency Test SAT(C_0)

Technical detail: the tableau algorithm

- works on a tree (semantics through viewing tree as an ABox):
  nodes represent elements of $\Delta^{\mathcal{I}}$, labelled with sub-concepts of C_0
  edges represent role-successorships between elements of $\Delta^{\mathcal{I}}$
- works on concepts in **negation normal form**: push negation inside using de Morgan' laws and

$$\neg(\exists R.C) \rightsquigarrow \forall R.\neg C \qquad \neg(\forall R.C) \rightsquigarrow \exists R.\neg C$$
$$\neg(\leq n\ R) \rightsquigarrow (\geq (n+1)R) \qquad \neg(\geq n\ R) \rightsquigarrow (\leq (n-1)R) \quad (n \geq 0)$$
$$\neg(\geq 0\ R) \rightsquigarrow A \sqcap \neg A$$

- is initialised with a tree consisting of a single (root) node x_0 with $\mathcal{L}(x_0) = \{C_0\}$:

$$x_0 \bullet \{C_0\}$$

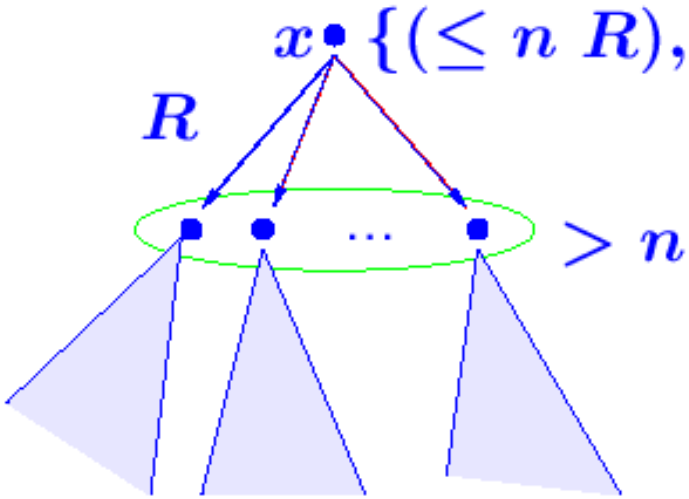
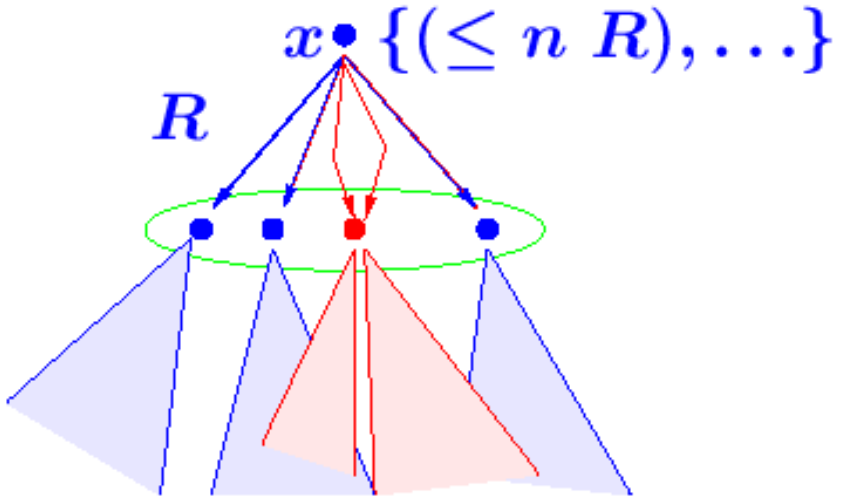
- a tree T contains a **clash** if, for a node x in T,

$$\{A, \neg A\} \subseteq \mathcal{L}(x) \text{ or}$$
$$\{(\geq m\ R), (\leq n\ R)\} \subseteq \mathcal{L}(x) \text{ for } n < m$$

ALC Tableau Rules

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcap C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_{\exists}$	$x \bullet \{\exists R.C, \ldots\}$ R $y \bullet \{C\}$
$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{\ldots\}$	$\rightarrow_{\forall}$	$x \bullet \{\forall R.C, \ldots\}$ R $y \bullet \{C, \ldots\}$

N Tableau Rules

$x\bullet\ \{(\geq n\ R), \ldots\}$ x has no R-succ.	$\rightarrow_{\geq}$	$x\bullet\ \{(\geq n\ R), \ldots\}$ R $y\bullet\ \{\}$
	$\rightarrow_{\leq}$	 merge two R-succs.

Soundness and Completeness (1)

Lemma

Let C_0 be an $\mathcal{ALCN}$ concept and T obtained by applying the tableau rules to C_0. Then

1. the rule application terminates,
2. if T is consistent and $\longrightarrow$ is applicable to T, then $\longrightarrow$ can be applied such that it yields consistent T',
3. if T contains a clash, then T has no model, and
4. if no more rules apply to T, then T defines (canonical) model for C_0.

Corollary

(1) The tableau algorithm is a PSpace decision procedure for consistency (and subsumption) of $\mathcal{ALCN}$ concepts

(2) $\mathcal{ALCN}$ has the tree model property

Soundness and Completeness (2)

Proof of the Lemma

1. (Termination) The algorithm "monotonically" constructs a tree whose
   depth is linear in $|C_0|$: quantifier depth decreases from node to succs.
   breadth is linear in $|C_0|$ (even if number in NRs are coded binarily)
2. (Local Consistency) Easy to prove (by definition of the semantics) that
   if $\mathcal{I}$ is a model of T, then $\longrightarrow$ can be applied to T such that
   $\mathcal{I}$ is a model of $\mathrm{T}' := \longrightarrow(\mathrm{T})$
3. Obvious: T with a clash has no model—recall definition of a clash:

$$\{A, \neg A\} \subseteq \mathcal{L}(x) \text{ or}$$
$$\{(\geq m\ R), (\leq n\ R)\} \subseteq \mathcal{L}(x) \text{ for } n < m$$

Soundness and Completeness (3)

Proof of the Lemma (ctd.)

4. (Canonical model) "Complete" tree $\mathbf{T}$ defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$
   edges define role-relationship
   $x \in A^{\mathcal{I}}$ iff $A \in \mathcal{L}(x)$ for concept names A

   Check that $C \in \mathcal{L}(x)$ implies $x \in C^{\mathcal{I}}$—if C is **no number restriction**.

   For NRs, if $(\geq n\ R) \in \mathcal{L}(x)$ and x has less than n R-successors,
   copy some R-successors (including sub-trees) to obtain n R-successors

   $\rightsquigarrow$ **canonical tree model for input concept**

Make the Tableau Algorithm Run in PSpace

To make the tableau algorithm run in PSpace:

Recall **Savitch**: PSpace = NPSpace

① observe that branches are independent from each other

② observe that each node (label) requires linear space only

③ recall that paths are of length $\leq |C_0|$

$\rightsquigarrow$ each path can be stored in $\mathcal{O}(|C_0|^2)$

④ construct/search the tree **depth first**

⑤ re-use space from already constructed branches

Extensibility (1)

This tableau algorithm can be modified to a PSpace decision procedure for

- ✔ $\mathcal{ALC}$ with **qualifying number restrictions**
  $(\geq n\ R\ C)$ and $(\leq n\ R\ C)$
- ✔ $\mathcal{ALC}$ with **inverse roles** (e.g. `has-child`$^-$)
- ✔ $\mathcal{ALC}$ with **role conjunction**
  $\exists(R \sqcap S).C$ and $\forall(R \sqcap S).C$
- ✔ **TBoxes with acyclic concept definitions** $A \doteq C$:

  unfolding (macro expansion) is easy, but suboptimal:
  may yield exponential blow-up

  lazy unfolding (unfolding on demand) is optimal, consistency in PSpace decidable

Extensibility (2)

Language extensions that require more elaborate techniques include

- **TBoxes with general axioms $C_i \sqsubseteq D_i$:**
  each node must be labelled with $\neg C_i \sqcup D_i$
  quantifier depth no longer decreases
  $\rightsquigarrow$ termination not guaranteed
- **Transitive closure of roles:**
  node labels $(\forall R^*.C)$ yields C in all R^n-successor labels
  quantifier depth no longer decreases
  $\rightsquigarrow$ termination not guaranteed

Use **blocking** (cycle detection) to ensure termination
(but the right blocking to not destroy soundness or completeness)

Algorithmic Optimisations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)

Dependency Directed Backtracking

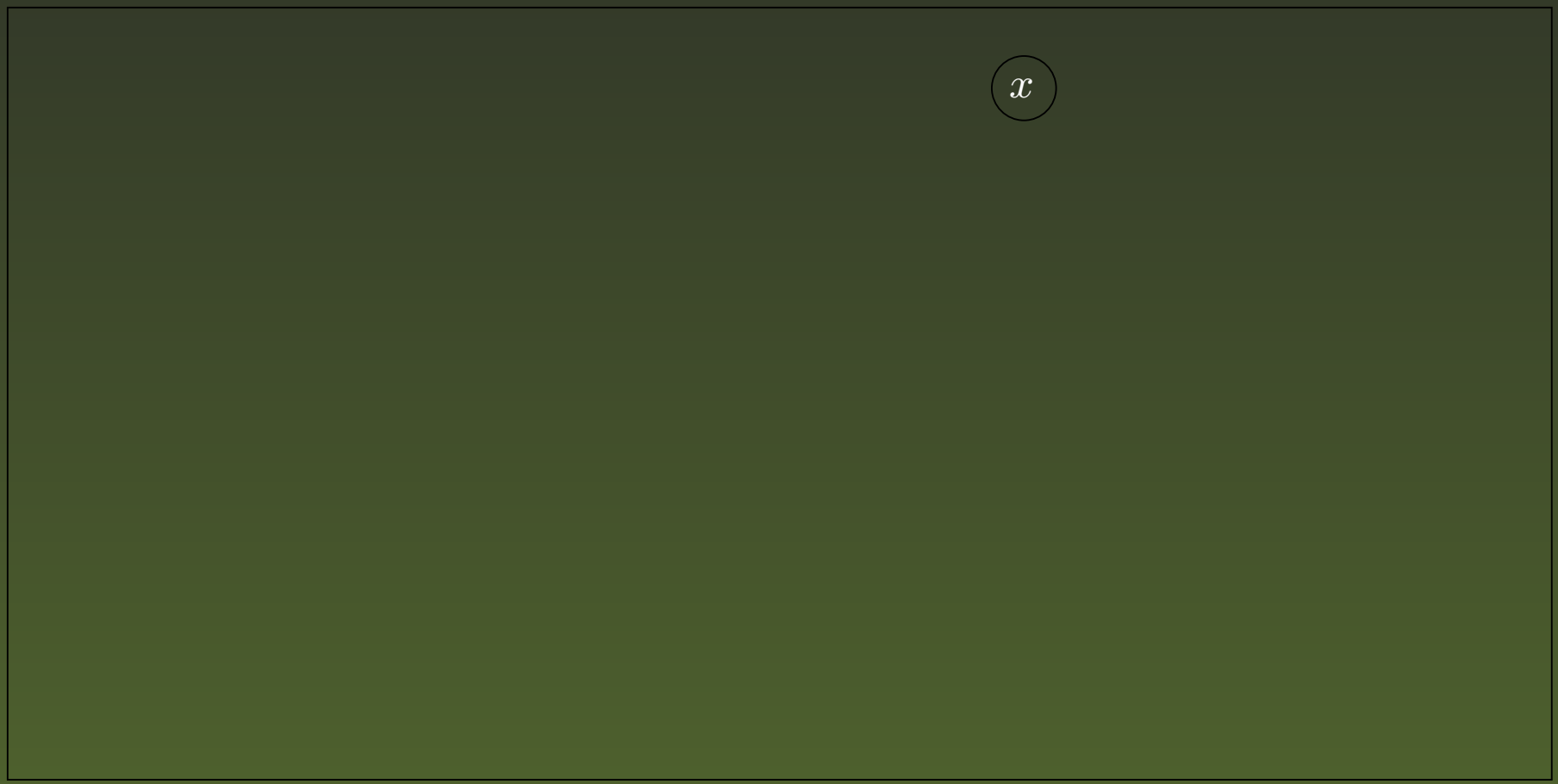
- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points **without exploring** alternative branches
  - Effect is to prune away part of the search space
  - Performance improvements with GALEN KB again **too large to measure**

Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$

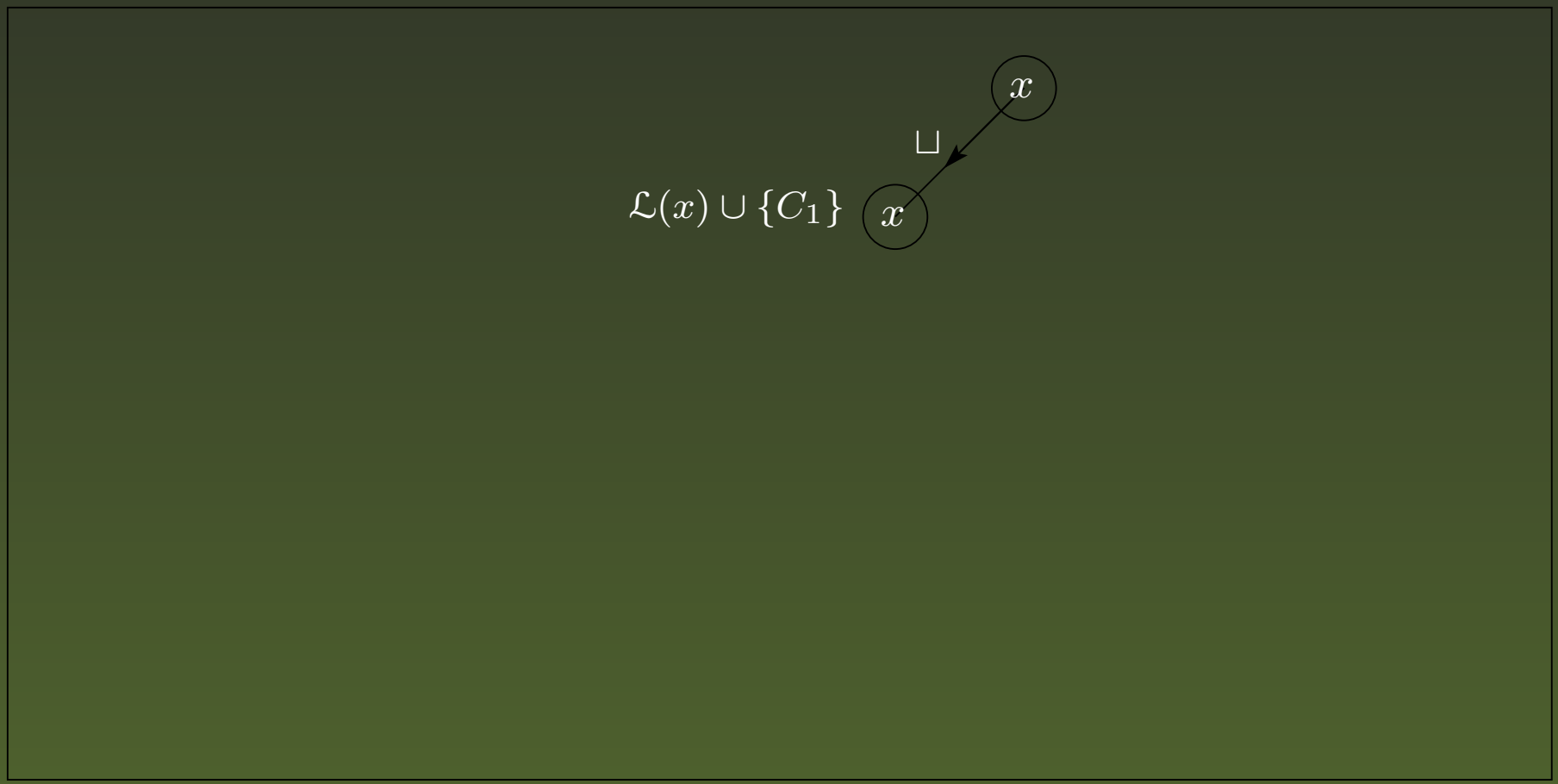
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



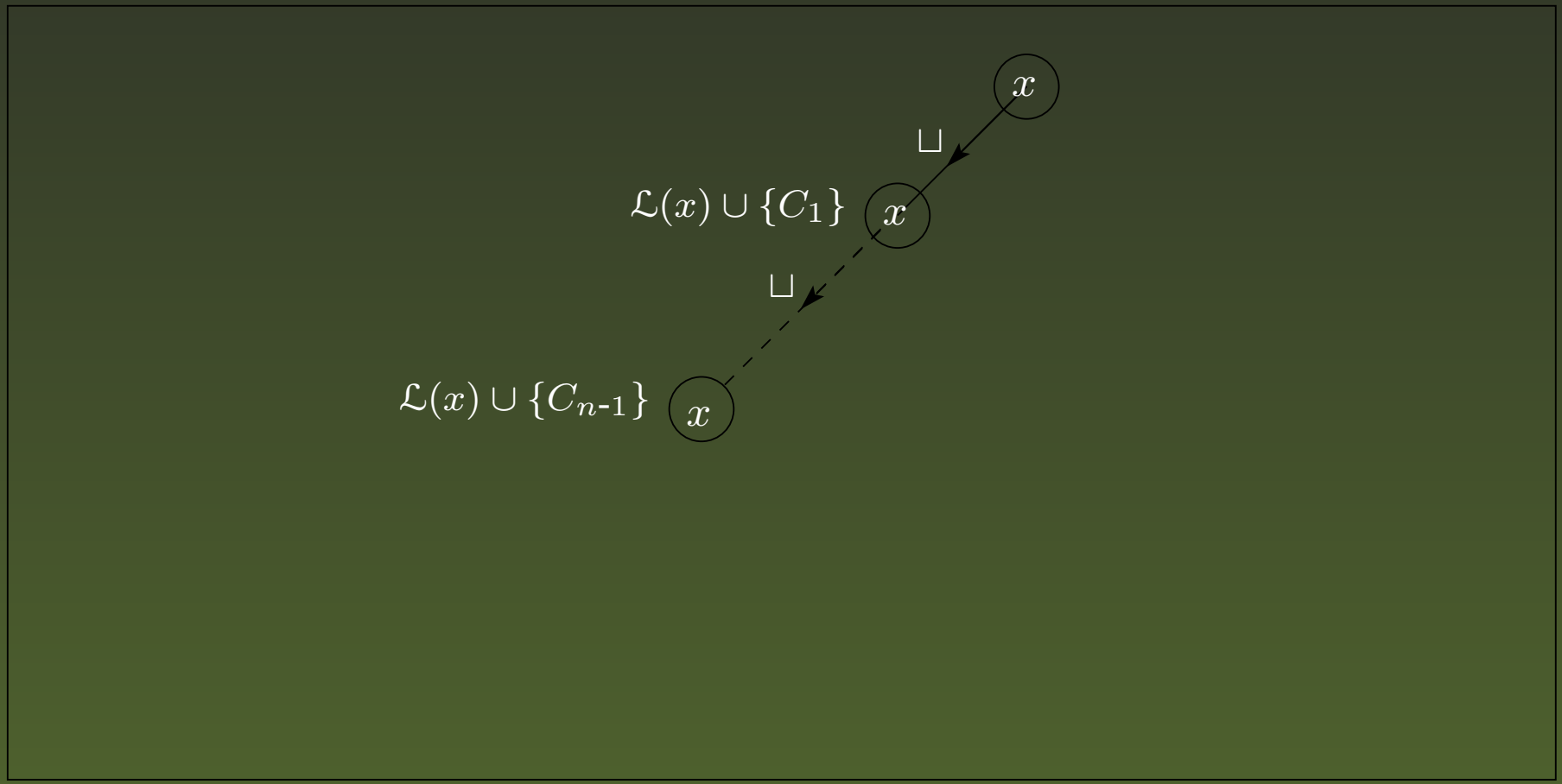
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



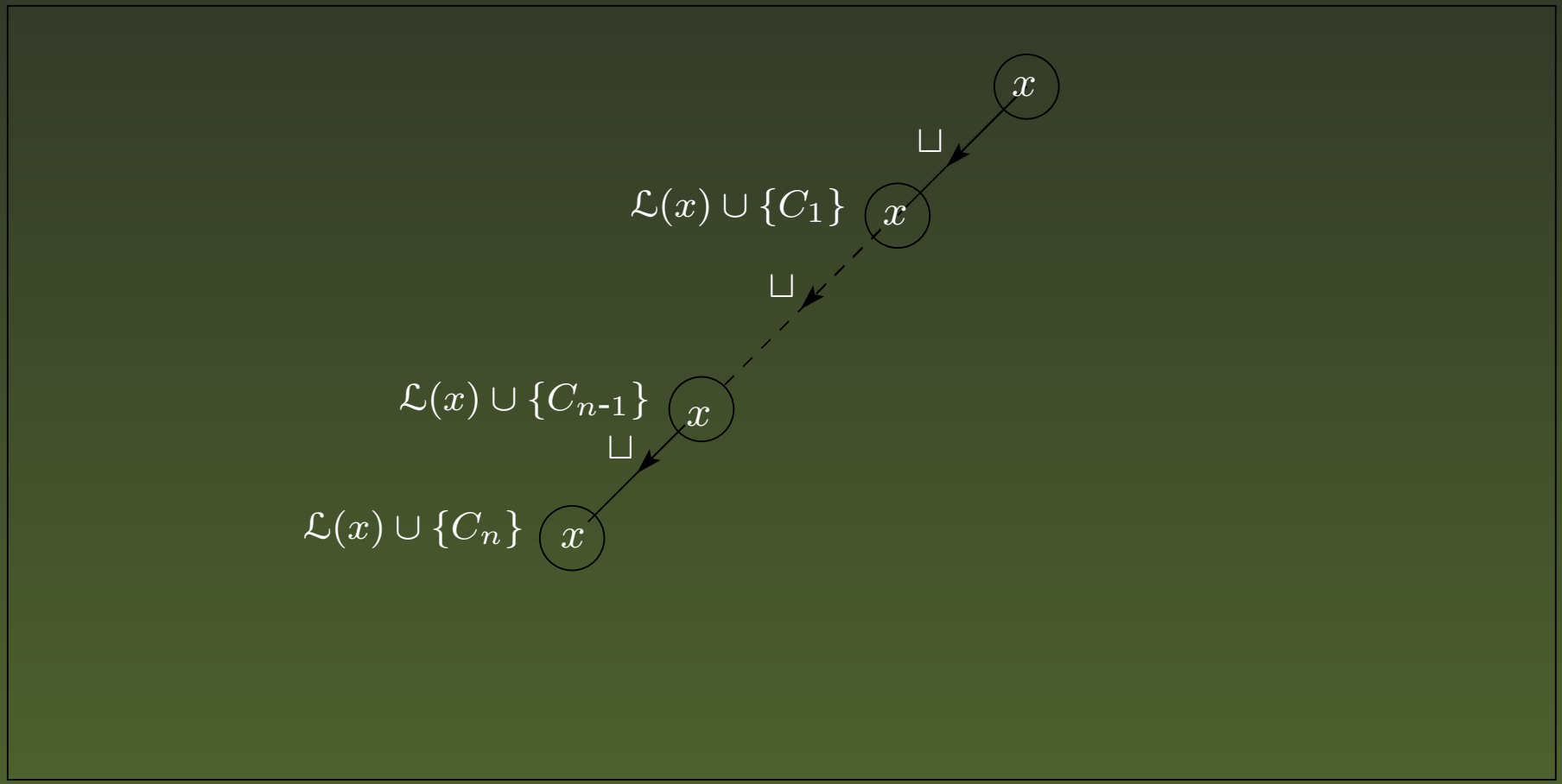
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



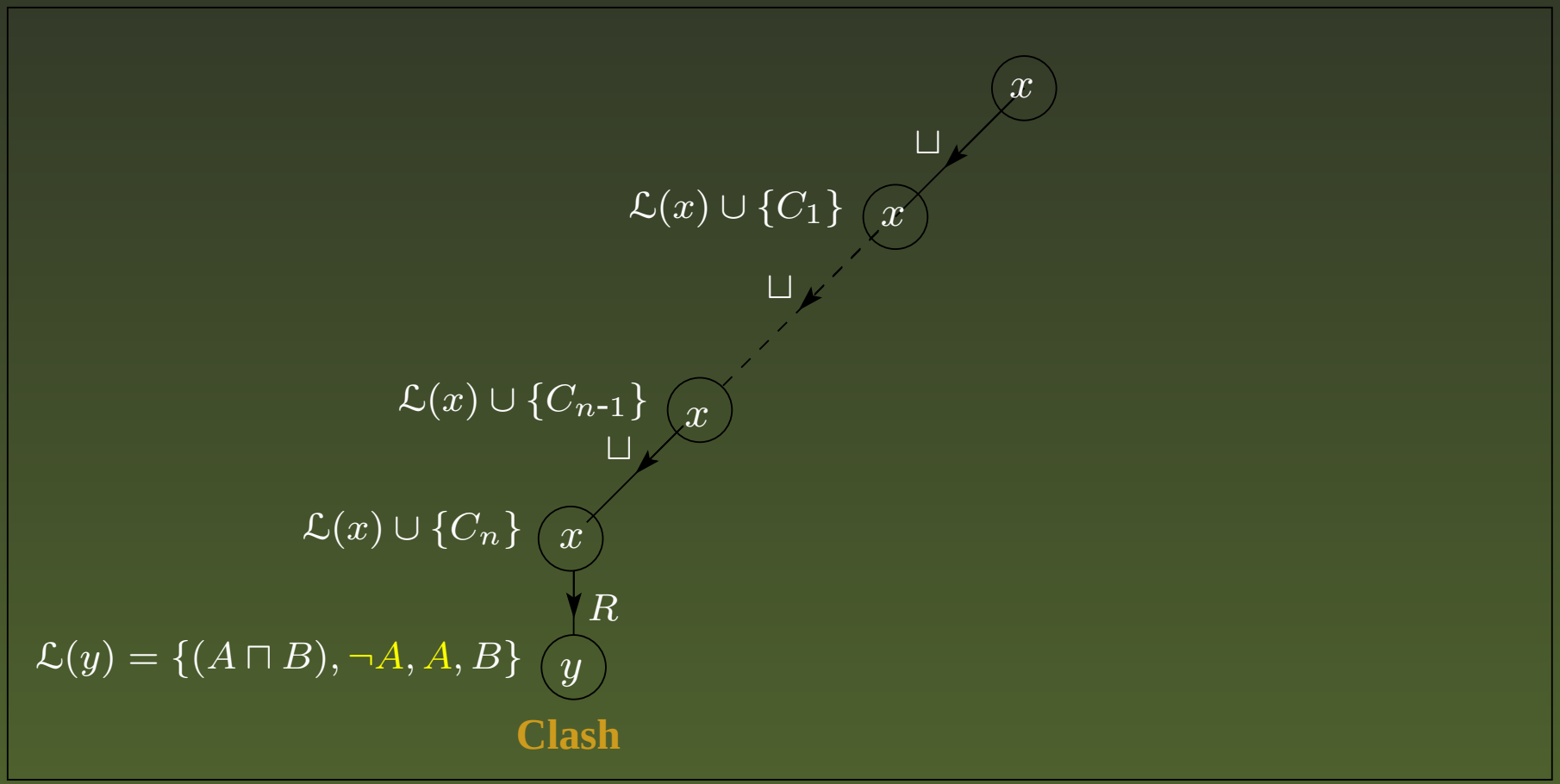
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



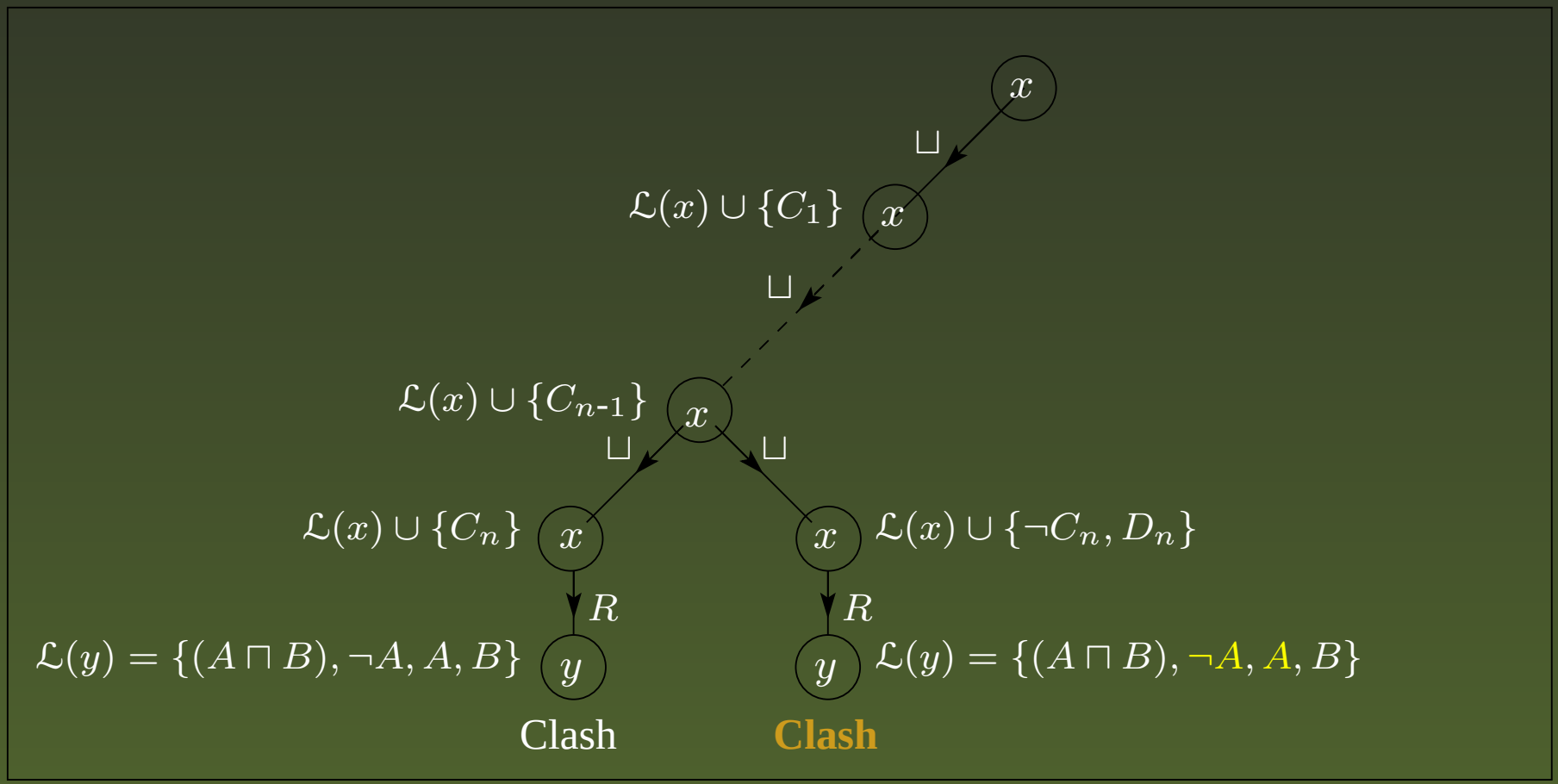
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



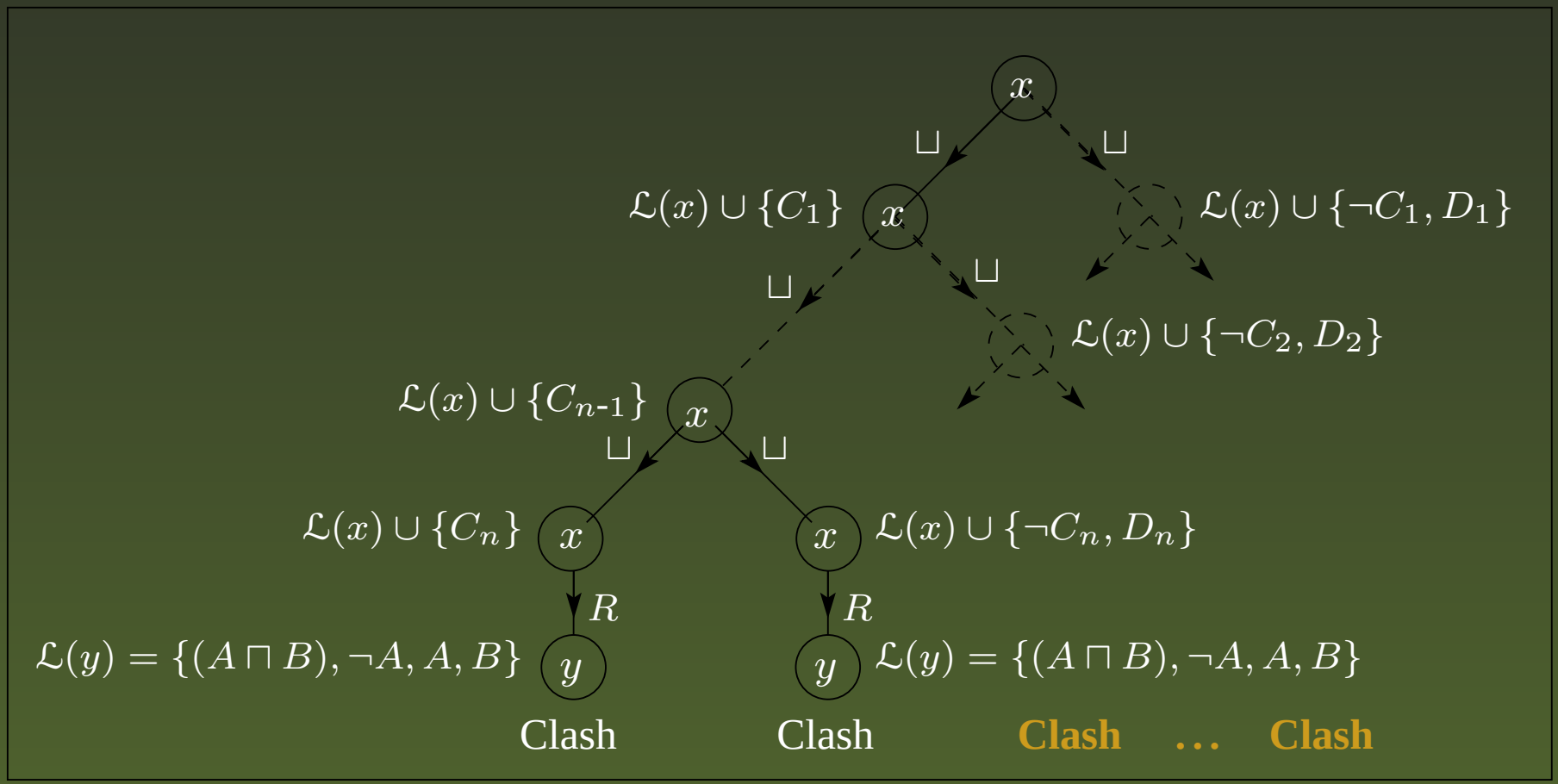
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



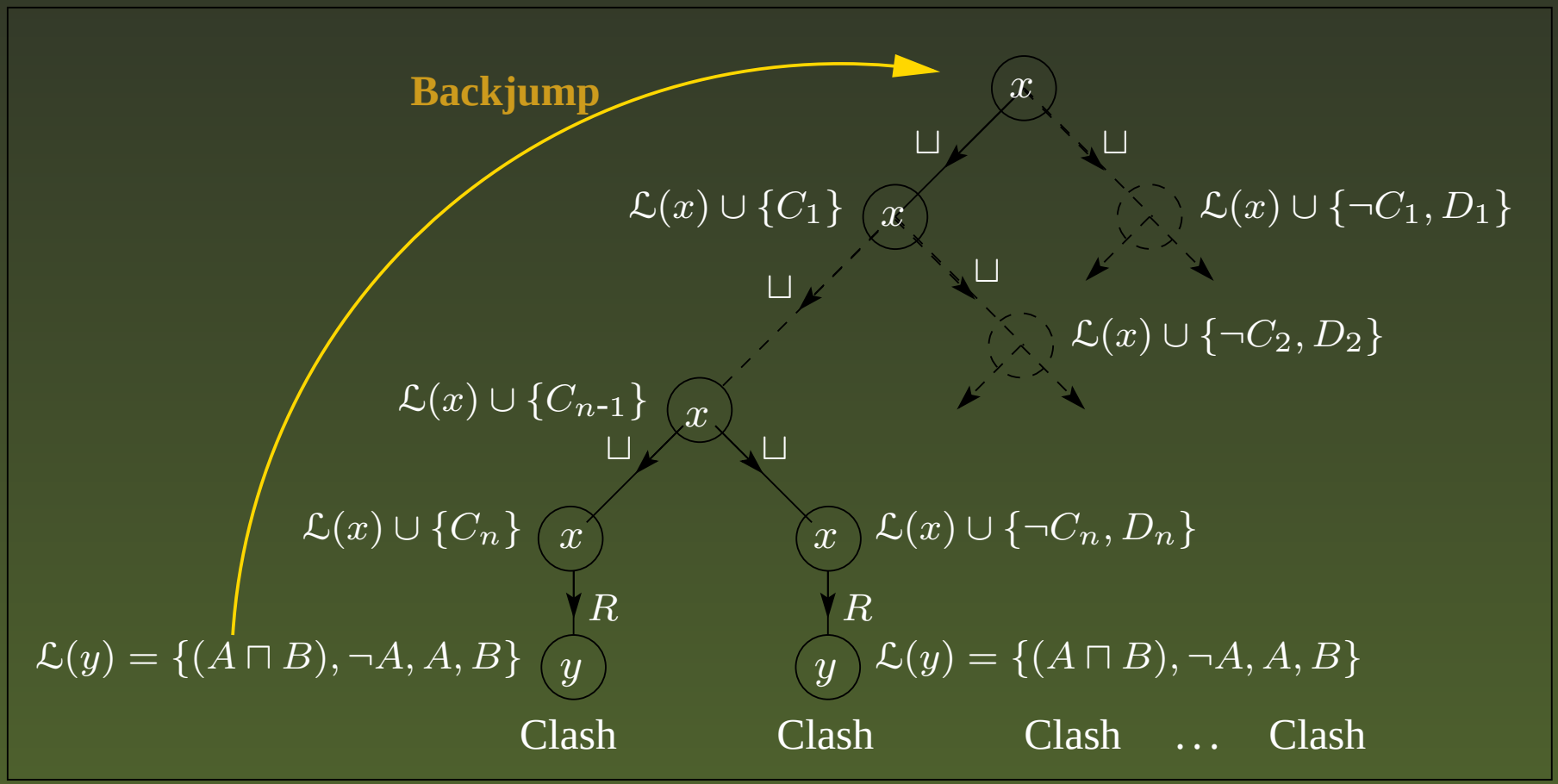
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



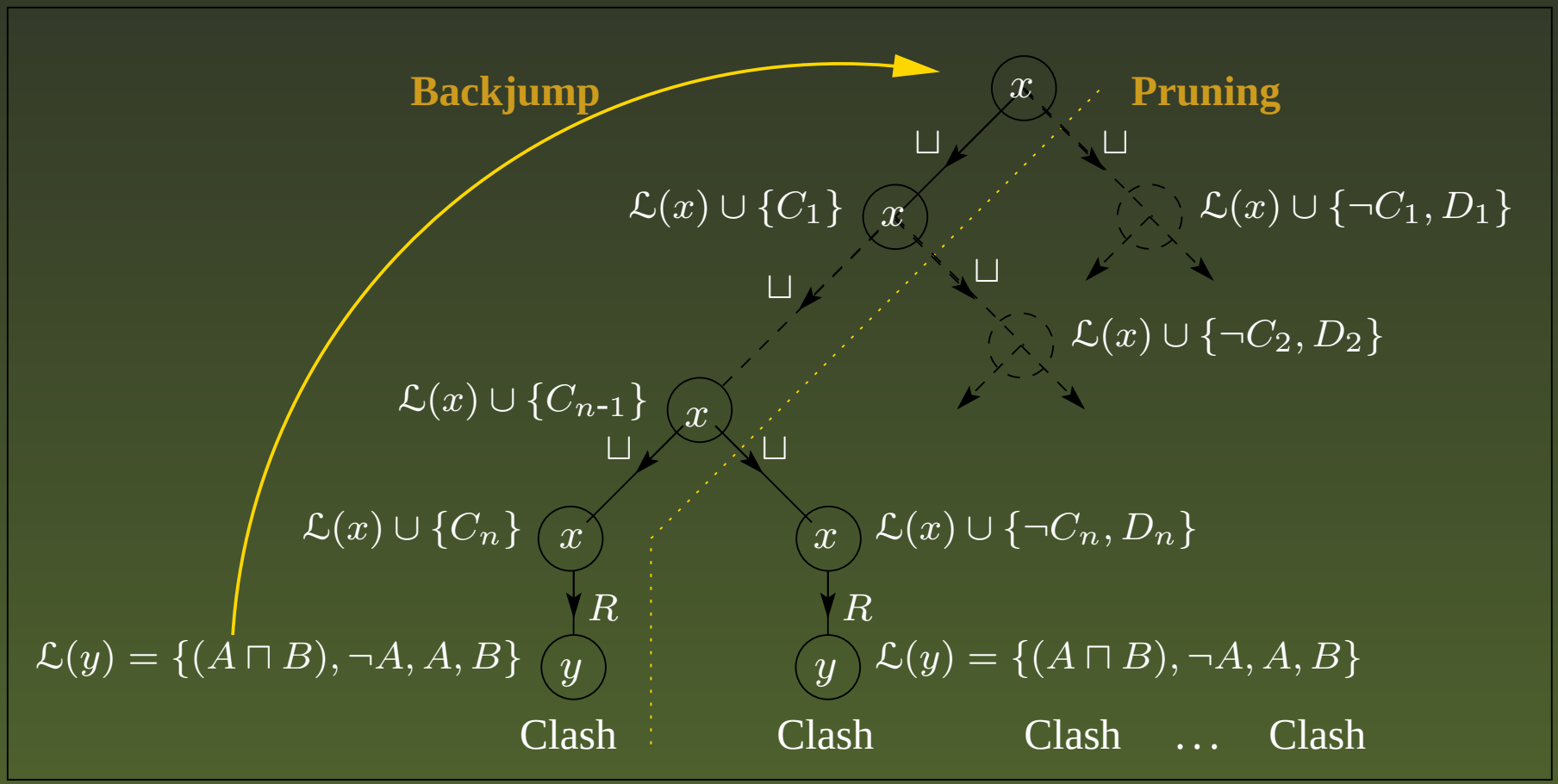
Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



Backjumping

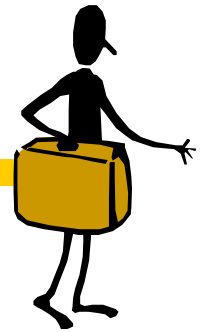
E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$



Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq D$ if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged
  - If not, continue with standard subsumption test
  - Can use same technique in sub-problems

Zusammenfassung, Kernpunkte



- Semantic Web (Teil 1)
- Algorithmus für ALCN-Konsistenztest
- Kernideen für Implementierung:
  - Dependency-directed Backtracking (Backjumping)
  - Caching

Was kommt beim nächsten Mal?

- Zeitliche und räumliche Zusammenhänge
- Qualitative Beziehungen:
  - Intervall-Relationen
    Topologie
- Constraint-Solving-Techniken für qualitative Zusammenhänge