NoCL 2015/16
Frame Your Reasoning: Introduction to the logic of Bunched Implications

Tadeusz Litak

January 18, 2016

Informatik 8, FAU Erlangen-Nürnberg

- Our next and last major system is once again a substructural logic . . .

- Our next and last major system is once again a substructural logic …
- …the logic of Bunched Implications (BI)

- Our next and last major system is once again a substructural logic ...
- ... the logic of Bunched Implications (BI)
- Several foundational papers by Peter O'Hearn and David Pym around 1999, especially *The Logic of Bunched Implications*, BSL 1999

- Our next and last major system is once again a substructural logic …
- …the logic of Bunched Implications (BI)
- Several foundational papers by Peter O'Hearn and David Pym around 1999, especially *The Logic of Bunched Implications*, BSL 1999
- Monograph by David Pym in 2002: *The Semantics and Proof Theory of the Logic of Bunched Implications*

- Our next and last major system is once again a substructural logic ...
- ... the logic of Bunched Implications (BI)
- Several foundational papers by Peter O'Hearn and David Pym around 1999, especially *The Logic of Bunched Implications*, BSL 1999
- Monograph by David Pym in 2002: *The Semantics and Proof Theory of the Logic of Bunched Implications*
- Also a paper *Possible worlds and resources: the semantics of* BI by Pym, O'Hearn and Yang, TCS 2004

- Our next and last major system is once again a substructural logic …
- … the logic of Bunched Implications (BI)
- Several foundational papers by Peter O'Hearn and David Pym around 1999, especially *The Logic of Bunched Implications*, BSL 1999
- Monograph by David Pym in 2002: *The Semantics and Proof Theory of the Logic of Bunched Implications*
- Also a paper *Possible worlds and resources: the semantics of* BI by Pym, O'Hearn and Yang, TCS 2004
- Important intelectual ancestor: John Reynolds

- The notion of *resource* central in the semantics for BI

- The notion of *resource* central in the semantics for BI
- Pym often quotes the definition of the notion from an early textbook on operating systems by B. Hansen (1973)

- The notion of *resource* central in the semantics for BI
- Pym often quotes the definition of the notion from an early textbook on operating systems by B. Hansen (1973)

> *The word "resource" covers physical components, processes, procedures and data structures; in short, any object referenced by computations.*

- The notion of *resource* central in the semantics for BI
- Pym often quotes the definition of the notion from an early textbook on operating systems by B. Hansen (1973)

> *The word "resource" covers physical components, processes, procedures and data structures; in short, any object referenced by computations.*

- Pym: the word "referenced" crucial. Resources can be shared by more than one computation

- The notion of *resource* central in the semantics for BI
- Pym often quotes the definition of the notion from an early textbook on operating systems by B. Hansen (1973)

> *The word "resource" covers physical components, processes, procedures and data structures; in short, any object referenced by computations.*

- Pym: the word "referenced" crucial. Resources can be shared by more than one computation
- This CS notion of resource is somewhat different to the one used in linear logic, which is tightly connected to *number-of-uses* interpretation

- The notion of *resource* central in the semantics for BI
- Pym often quotes the definition of the notion from an early textbook on operating systems by B. Hansen (1973)

> *The word "resource" covers physical components, processes, procedures and data structures; in short, any object referenced by computations.*

- Pym: the word "referenced" crucial. Resources can be shared by more than one computation
- This CS notion of resource is somewhat different to the one used in linear logic, which is tightly connected to *number-of-uses* interpretation
- Correspondingly, BI has quite different formal properties

- This interpretation central to the success story of BI in CS

- This interpretation central to the success story of BI in CS
- A starting point: Ishtiaq, O'Hearn (POPL 2001) BI *as an assertion language for mutable data structures*

- This interpretation central to the success story of BI in CS
- A starting point: Ishtiaq, O'Hearn (POPL 2001) BI *as an assertion language for mutable data structures*
- Assertional core for a Hoare logic promoted by John Reynolds for reasoning about *shared mutable data structures*

- This interpretation central to the success story of BI in CS
- A starting point: Ishtiaq, O'Hearn (POPL 2001) BI *as an assertion language for mutable data structures*
- Assertional core for a Hoare logic promoted by John Reynolds for reasoning about *shared mutable data structures*
- Widely known as *separation logic*

- This interpretation central to the success story of BI in CS
- A starting point: Ishtiaq, O'Hearn (POPL 2001) BI *as an assertion language for mutable data structures*
- Assertional core for a Hoare logic promoted by John Reynolds for reasoning about *shared mutable data structures*
- Widely known as *separation logic*
- We will spend now quite a few slides on motivation

- The definition of shared mutable data structures by John Reynolds: *structures where an updatable field can be referenced from more than one point*

- The definition of shared mutable data structures by John Reynolds: *structures where an updatable field can be referenced from more than one point*
- Lists (single- or doubly-linked), queues, trees, stacks …

- The definition of shared mutable data structures by John Reynolds: *structures where an updatable field can be referenced from more than one point*
- Lists (single- or doubly-linked), queues, trees, stacks …
- … and with these, issues of pointers, heaps, allocation …

- The definition of shared mutable data structures by John Reynolds: *structures where an updatable field can be referenced from more than one point*
- Lists (single- or doubly-linked), queues, trees, stacks ...
- ... and with these, issues of pointers, heaps, allocation ...
- Clearly, a subject of immense significance

- The definition of shared mutable data structures by John Reynolds: *structures where an updatable field can be referenced from more than one point*
- Lists (single- or doubly-linked), queues, trees, stacks . . .
- . . . and with these, issues of pointers, heaps, allocation . . .
- Clearly, a subject of immense significance
- Also quite clearly, a major source of errors, issues and bugs: dangling pointers, memory leaks, segmentation faults . . .

Shared mutable data structures: whence the pain?

- Perhaps unsurprisingly: such programs notoriously difficult to reason about

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*
    - Microsoft's SLAM Protocol (mentioned in Bill Gates' 2002 keynote address): properties of procedure calls in device drivers, e.g. *any call to* `ReleaseSpinLock` *is preceded by a call to* `AquireSpinLock`

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*
  - Microsoft's SLAM Protocol (mentioned in Bill Gates' 2002 keynote address): properties of procedure calls in device drivers, e.g. *any call to* `ReleaseSpinLock` *is preceded by a call to* `AquireSpinLock`
  - In Nov. 2003, the Astrée static analyzer proved *completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system*

    see the project webpage

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*
    - Microsoft's SLAM Protocol (mentioned in Bill Gates' 2002 keynote address): properties of procedure calls in device drivers, e.g. *any call to* `ReleaseSpinLock` *is preceded by a call to* `AquireSpinLock`
    - In Nov. 2003, the Astrée static analyzer proved *completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system*

      see the project webpage
- ... but even these projects were steering clear of automatic heap verification!

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*
    - Microsoft's SLAM Protocol (mentioned in Bill Gates' 2002 keynote address): properties of procedure calls in device drivers, e.g. *any call to* `ReleaseSpinLock` *is preceded by a call to* `AquireSpinLock`
    - In Nov. 2003, the Astrée static analyzer proved *completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system*
      see the project webpage
- ... but even these projects were steering clear of automatic heap verification!
    - The first assumed memory safety

- Perhaps unsurprisingly: such programs notoriously difficult to reason about
- Examples taken from Peter O'Hearn presentations: by 2000s, *impressive practical advances in automatic program verification*
  - Microsoft's SLAM Protocol (mentioned in Bill Gates' 2002 keynote address): properties of procedure calls in device drivers, e.g. *any call to* `ReleaseSpinLock` *is preceded by a call to* `AquireSpinLock`
  - In Nov. 2003, the Astrée static analyzer proved *completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system*

    see the project webpage
- . . . but even these projects were steering clear of automatic heap verification!
  - The first assumed memory safety
  - The second assumed no dynamic pointer allocation

- And why (shared) mutable data structures are so problematic?

- And why (shared) mutable data structures are so problematic?
- Let us quote from O'Hearn, Reynolds and Yang:

- And why (shared) mutable data structures are so problematic?
- Let us quote from O'Hearn, Reynolds and Yang:
- *The main difficulty is not one of finding an in-principle adequate axiomatization of pointer operations ...*

- And why (shared) mutable data structures are so problematic?
- Let us quote from O'Hearn, Reynolds and Yang:
- *The main difficulty is not one of finding an in-principle adequate axiomatization of pointer operations . . .*
- *rather there is a mismatch between simple intuitions about the way that pointer operations work and the complexity of their axiomatic treatments . . .*

- And why (shared) mutable data structures are so problematic?
- Let us quote from O'Hearn, Reynolds and Yang:
- *The main difficulty is not one of finding an in-principle adequate axiomatization of pointer operations ...*
- *rather there is a mismatch between simple intuitions about the way that pointer operations work and the complexity of their axiomatic treatments ...*
- *For example, pointer assignment is operationally simple, but when there is aliasing, arising from several pointers to a given cell, then an alteration to that cell may affect the values of many syntactically unrelated expressions ...*

- Consider the following code:

```
PROC appendlist(x,y)
  LOCAL t, u;
  IF (x == nil) THEN x := y ELSE
    t := x; u := t ->n;
    WHILE not (u == nil) DO t := u; u := t ->n END;
    t ->n := y
  ENDIF
ENDPROC
```

- Consider the following code:

```
PROC appendlist(x,y)
  LOCAL t, u;
  IF (x == nil) THEN x := y ELSE
    t := x; u := t ->n;
    WHILE not (u == nil) DO t := u; u := t ->n END;
    t ->n := y
  ENDIF
ENDPROC
```

- Assume the assertion language contains a predicate `ls (x, t)` meaning *a linked list segment*: there is a path from `x` to `t` and ($x \neq t$ or $t = nil$)

- Consider the following code:

```
PROC appendlist(x,y)
  LOCAL t, u;
  IF (x == nil) THEN x := y ELSE
    t := x; u := t ->n;
    WHILE not (u == nil) DO t := u; u := t ->n END;
    t ->n := y
  ENDIF
ENDPROC
```

- Assume the assertion language contains a predicate `ls (x, t)` meaning *a linked list segment*: there is a path from `x` to `t` and (`x` $\neq$ `t` or `t` $=$ `nil`)
- A complete linked list: `ls (x, nil)`

- Consider the following code:

```
PROC appendlist(x,y)
  LOCAL t, u;
  IF (x == nil) THEN x := y ELSE
    t := x; u := t ->n;
    WHILE not (u == nil) DO t := u; u := t ->n END;
    t ->n := y
  ENDIF
ENDPROC
```

- Assume the assertion language contains a predicate ls (x, t) meaning *a linked list segment*: there is a path from x to t and ($x \neq t$ or $t = nil$)
- A complete linked list: ls (x, nil)
- Is this a valid triple?

```
{ls (x, nil) and ls(y, nil)}
 appendlist(x,y)
{ls (x, nil)}
```

- `x` cannot be a sublist of `y` : we have to be able to state that

- `x` cannot be a sublist of `y` : we have to be able to state that
- but also, `y` cannot be a sublist of `x`. Is this enough?

- `x` cannot be a sublist of `y` : we have to be able to state that
- but also, `y` cannot be a sublist of `x`. Is this enough?
- `x` and `y` should not have a common final segment . . .

- `x` cannot be a sublist of `y` : we have to be able to state that
- but also, `y` cannot be a sublist of `x`. Is this enough?
- `x` and `y` should not have a common final segment …
- …in short, they should operate in disjoint areas of memory

- `x` cannot be a sublist of `y` : we have to be able to state that
- but also, `y` cannot be a sublist of `x`. Is this enough?
- `x` and `y` should not have a common final segment . . .
- . . . in short, they should operate in disjoint areas of memory
- And now think of the loop invariants

- `x` cannot be a sublist of `y` : we have to be able to state that
- but also, `y` cannot be a sublist of `x`. Is this enough?
- `x` and `y` should not have a common final segment ...
- ...in short, they should operate in disjoint areas of memory
- And now think of the loop invariants
- And remember you should not only be able to state all the information, but have some way to reason about, decide and infer such assertions

- And now try to reason about a bigger program using `appendlist`

- And now try to reason about a bigger program using `appendlist`
- It can use many other data structures and areas of the heap

- And now try to reason about a bigger program using `appendlist`
- It can use many other data structures and areas of the heap
- When `appendlist` is invoked, the Floyd-Hoare reasoning can only use information in the contract

- And now try to reason about a bigger program using `appendlist`
- It can use many other data structures and areas of the heap
- When `appendlist` is invoked, the Floyd-Hoare reasoning can only use information in the contract
- Pre- and postconditions should allow us to infer that nothing outside the area occupied now by `x` changed ...

- And now try to reason about a bigger program using `appendlist`
- It can use many other data structures and areas of the heap
- When `appendlist` is invoked, the Floyd-Hoare reasoning can only use information in the contract
- Pre- and postconditions should allow us to infer that nothing outside the area occupied now by `x` changed ...
- In short, we are staring in the face of ...

The frame problem

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence
- McCarthy and Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence*, 1969

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence
- McCarthy and Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence*, 1969
- Try to formalize, say, the obvious fact that P can get into conversation with Q by looking up Q's number in the phone book and then dialling it up

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence
- McCarthy and Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence*, 1969
- Try to formalize, say, the obvious fact that P can get into conversation with Q by looking up Q's number in the phone book and then dialling it up
- Whatever formalization you are going to write, it is almost certain in the end you will be missing some hypotheses like *if a person has a telephone he still has it after looking up a number in the telephone book ...*

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence
- McCarthy and Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence*, 1969
- Try to formalize, say, the obvious fact that P can get into conversation with Q by looking up Q's number in the phone book and then dialling it up
- Whatever formalization you are going to write, it is almost certain in the end you will be missing some hypotheses like *if a person has a telephone he still has it after looking up a number in the telephone book ...*
- ... or that *if P looks up Q's phone-number in the book, he will know it*

- The term the frame problem appeared in early days of (the philosophy of) Artificial Intelligence
- McCarthy and Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence*, 1969
- Try to formalize, say, the obvious fact that P can get into conversation with Q by looking up Q's number in the phone book and then dialling it up
- Whatever formalization you are going to write, it is almost certain in the end you will be missing some hypotheses like *if a person has a telephone he still has it after looking up a number in the telephone book* ...
- ... or that *if P looks up Q's phone-number in the book, he will know it*
- And then of course there are all sorts of special-case scenarios you need to exclude. Quoting McCarthy and Hayes still further ...

- *The page with Q's number may be torn out.*

- *The page with Q's number may be torn out.*
- *P may be blind.*

- *The page with Q's number may be torn out.*
- *P may be blind.*
- *Someone may have deliberately inked out Qs number.*

- *The page with Q's number may be torn out.*
- *P may be blind.*
- *Someone may have deliberately inked out Qs number.*
- *The telephone company may have made the entry incorrectly.*

- *The page with Q's number may be torn out.*
- *P may be blind.*
- *Someone may have deliberately inked out Qs number.*
- *The telephone company may have made the entry incorrectly.*
- *Q may have got the telephone only recently.*

- *The page with Q's number may be torn out.*
- *P may be blind.*
- *Someone may have deliberately inked out Qs number.*
- *The telephone company may have made the entry incorrectly.*
- *Q may have got the telephone only recently.*
- *The phone system may be out of order.*

- *The page with Q's number may be torn out.*
- *P may be blind.*
- *Someone may have deliberately inked out Qs number.*
- *The telephone company may have made the entry incorrectly.*
- *Q may have got the telephone only recently.*
- *The phone system may be out of order.*
- *Q may be incapacitated suddenly . . .*

- *When formally describing a change in a system, how do we specify what parts of the state of the system are not affected by that change?*

  as paraphrased later by Kassios

- *When formally describing a change in a system, how do we specify what parts of the state of the system are not affected by that change?*

  as paraphrased later by Kassios

- The fact that an analogous problem arises with formal specifications using Floyd-Hoare logics discussed (in the context of OO code: inheritance issues etc.) by

  Borgida, Mylopoulos and Reiter. On the frame problem in procedure specifications. IEEE Transactions of Software Engineering, 1995

- *When formally describing a change in a system, how do we specify what parts of the state of the system are not affected by that change?*

  as paraphrased later by Kassios

- The fact that an analogous problem arises with formal specifications using Floyd-Hoare logics discussed (in the context of OO code: inheritance issues etc.) by

  Borgida, Mylopoulos and Reiter. On the frame problem in procedure specifications. IEEE Transactions of Software Engineering, 1995

- Hoare-style formalisms invented in the noughties for shared mutable data structures use the term *frame* very prominently . . .

- In separation logic (Reynolds, Ishtiaq, O'Hearn ..., 1999–2002 and developed since), a central Hoare rule (proposed by O'Hearn) is the frame rule

  It has to be either assumed as an axiom or derived.

  Separation logic also found to be useful in the context of concurrency and other forms of resource sharing. Extended with *abstract predicates* by Parkinson and Bierman 2005

- In separation logic (Reynolds, Ishtiaq, O'Hearn . . . , 1999–2002 and developed since), a central Hoare rule (proposed by O'Hearn) is the frame rule

  It has to be either assumed as an axiom or derived.

  Separation logic also found to be useful in the context of concurrency and other forms of resource sharing. Extended with *abstract predicates* by Parkinson and Bierman 2005

- An alternative approach proposed by Kassios in 2006: dynamic frames. A later relative: implict dynamic frames (Smans, Jacobs, Piessens 2009)

  Dynamic frames intended to use also in the OO context. Implicit dynamic frames are closer in spirit to separation logic

- A central idea of separation logic: suspend thinking of the global heap when writing/reading specifications

- A central idea of separation logic: suspend thinking of the global heap when writing/reading specifications
- Quoting Berdine, Calcagno, O'Hearn: think of heaplets, portions of heap.

- A central idea of separation logic: suspend thinking of the global heap when writing/reading specifications
- Quoting Berdine, Calcagno, O'Hearn: think of heaplets, portions of heap.
- a spec {P} C {Q} says that *if C is given a heaplet satisfying P then it will never try to access heap outside of P (other than cells allocated during execution) and it will deliver a heaplet satisfying Q if it terminates*

- A central idea of separation logic: suspend thinking of the global heap when writing/reading specifications
- Quoting Berdine, Calcagno, O'Hearn: think of heaplets, portions of heap.
- a spec {P} C {Q} says that *if C is given a heaplet satisfying P then it will never try to access heap outside of P (other than cells allocated during execution) and it will deliver a heaplet satisfying Q if it terminates*
- (Of course, this has implications for how C acts on the global heap.)

  In the dynamic frames approach, one does think in terms of global heap, using instead: "reads/modifies" clauses in assertions, "swinging pivot postconditions" and permission masks

- Assertions about disjoint heaplets are combined using the spatial conjunction $A_1 * A_2 \ldots$

- Assertions about disjoint heaplets are combined using the spatial conjunction $A_1 * A_2 \dots$
- …a.k.a. the separating conjunction or the independent conjunction

- Assertions about disjoint heaplets are combined using the spatial conjunction $A_1 * A_2$ …
- …a.k.a. the separating conjunction or the independent conjunction
- As you can already see, for substructural logicians it's a special case of fusion or multiplicative conjunction

- Assertions about disjoint heaplets are combined using the spatial conjunction $A_1 * A_2 \ldots$
- …a.k.a. the separating conjunction or the independent conjunction
- As you can already see, for substructural logicians it's a special case of fusion or multiplicative conjunction
- Reynolds was rather inspired by an early work of Burstall:

  Some techniques for proving correctness of programs which alter data structures, 1972

- Recall the problem with

```
{ls(x, nil) and ls(y, nil)}
 appendlist(x,y)
{ls(x, nil)}
```

- Recall the problem with

```
{ls(x, nil) and ls(y, nil)}
 appendlist(x,y)
{ls(x, nil)}
```

- This is looking much better:

```
{ls(x, nil) * ls(y, nil)}
 appendlist(x,y)
{ls(x, nil)}
```

- We can also state the frame rule:

$$\frac{\{A\}C\{B\} \quad \text{no variable free in } A' \text{ modified by } C}{\{A * A'\}C\{B * A'\}}$$

- We can also state the frame rule:

$$\frac{\{A\}C\{B\} \quad \text{no variable free in } A' \text{ modified by } C}{\{A * A'\}C\{B * A'\}}$$

- The second premise, of course, is suitably formalized

- BI has also other connectives, like the the magic wand $-\!\!*$

- BI has also other connectives, like the the magic wand $-\!\!*$
- This is precisely multiplicative implication in the BI setting

- BI has also other connectives, like the the magic wand $-\!\!*$
- This is precisely multiplicative implication in the BI setting
- It is useful, e.g., for deriving weakest preconditions

  one as a rule uses some form of implication in weakest preconditions

- BI has also other connectives, like the the magic wand $-\!\!*$
- This is precisely multiplicative implication in the BI setting
- It is useful, e.g., for deriving weakest preconditions

  one as a rule uses some form of implication in weakest preconditions
- However, BI unlike linear logic has also additive implication

  ...which is precisely the implication of intuitionistic logic

- Thus, BI combines intuitionistic logic for additive connectives …

  $\wedge, \vee, \rightarrow, \top, \bot$

- Thus, BI combines intuitionistic logic for additive connectives ...

  $\wedge, \vee, \rightarrow, \top, \bot$

- ... with multiplicative connectives of linear logic:

  $*$, $-\!*$, 1

  Note the absence of multiplicative 0 or disjunction

  Neither absence is coincidental. It's possible to combine BI with classical linear logic, but this kills the most intended semantics

- Thus, BI combines intuitionistic logic for additive connectives …

  $\wedge, \vee, \rightarrow, \top, \bot$

- … with multiplicative connectives of linear logic:

  $*$, $-\!*$, 1

  Note the absence of multiplicative 0 or disjunction

  Neither absence is coincidental. It's possible to combine BI with classical linear logic, but this kills the most intended semantics

- Would be straightforward to define semantically if we did category theory …

  Bicartesian DCC's (doubly closed categories): combining two monoidal closed structures, one of which is bicartesian

- Thus, BI combines intuitionistic logic for additive connectives . . .
  $\wedge, \vee, \rightarrow, \top, \bot$
- . . . with multiplicative connectives of linear logic:
  $*$, $-\!*$, 1

  Note the absence of multiplicative 0 or disjunction

  Neither absence is coincidental. It's possible to combine BI with classical linear logic, but this kills the most intended semantics
- Would be straightforward to define semantically if we did category theory . . .

  Bicartesian DCC's (doubly closed categories): combining two monoidal closed structures, one of which is bicartesian
- . . . or at least algebra

  Heyting algebras equipped with an additional structure of *residuated commutative monoid*: could call these BI-algebras

- What we are going to see instead is a simplified Kripke semantics: *preordered resource monoids*

- What we are going to see instead is a simplified Kripke semantics: *preordered resource monoids*
- But we will begin with proof-theoretical approach

  much like in the linear logic case

- What we are going to see instead is a simplified Kripke semantics: *preordered resource monoids*
- But we will begin with proof-theoretical approach

  much like in the linear logic case
- There are some obvious difficulties here though

- What we are going to see instead is a simplified Kripke semantics: *preordered resource monoids*
- But we will begin with proof-theoretical approach

  much like in the linear logic case
- There are some obvious difficulties here though
- For example, as we have already learned, having the additive implication forces distributivity laws for additive connectives: how does the system reflect that without collapsing multiplicatives?

- What we are going to see instead is a simplified Kripke semantics: *preordered resource monoids*
- But we will begin with proof-theoretical approach

  much like in the linear logic case
- There are some obvious difficulties here though
- For example, as we have already learned, having the additive implication forces distributivity laws for additive connectives: how does the system reflect that without collapsing multiplicatives?
- More basically and prosaically, how do we distinguish introduction rules for multiplicative and additive implication?

- A solution found in proof theory of earlier substructural logics with distributive additives

- A solution found in proof theory of earlier substructural logics with distributive additives
- Comma on the left plays the role of multiplicative conjunction?

- A solution found in proof theory of earlier substructural logics with distributive additives
- Comma on the left plays the role of multiplicative conjunction?
- Well, introduce another symbol — say, ; — which will play the same role wrt additive conjunction

- A solution found in proof theory of earlier substructural logics with distributive additives
- Comma on the left plays the role of multiplicative conjunction?
- Well, introduce another symbol — say, ; — which will play the same role wrt additive conjunction
- Then we have two different introduction rules we wanted:

$$\frac{\Gamma, \phi \Rightarrow \psi}{\Gamma \Rightarrow \phi \mathrel{-\!\!*} \psi} \quad \text{vs.} \quad \frac{\Gamma; \phi \Rightarrow \psi}{\Gamma \Rightarrow \phi \to \psi}$$

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree
- Internal nodes are labelled with semicolons and commas

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree
- Internal nodes are labelled with semicolons and commas
- Leafs are labelled with actual formulas

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree
- Internal nodes are labelled with semicolons and commas
- Leafs are labelled with actual formulas
- Such beasts have long been known to another tribe of substructural logicians: relevant logicians ...

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree
- Internal nodes are labelled with semicolons and commas
- Leafs are labelled with actual formulas
- Such beasts have long been known to another tribe of substructural logicians: relevant logicians . . .
- . . . under the name of bunches

- This means that the left side of $\Rightarrow$ is no longer just a list or a multiset
- It is a rather special tree
- Internal nodes are labelled with semicolons and commas
- Leafs are labelled with actual formulas
- Such beasts have long been known to another tribe of substructural logicians: relevant logicians ...
- ...under the name of bunches
- Yes, this is where the name of BI comes from

- Note: even the empty antecedent on the left must get split into multiplicative and additive emptiness: $\emptyset_m$ vs. $\emptyset_a$

- Note: even the empty antecedent on the left must get split into multiplicative and additive emptiness: $\emptyset_m$ vs. $\emptyset_a$
- First corresponds to 1, the other to $\top$

- Note: even the empty antecedent on the left must get split into multiplicative and additive emptiness: $\emptyset_m$ vs. $\emptyset_a$
- First corresponds to 1, the other to $\top$
- We have reached the stage where slides are no longer useful

- Note: even the empty antecedent on the left must get split into multiplicative and additive emptiness: $\emptyset_m$ vs. $\emptyset_a$
- First corresponds to 1, the other to $\top$
- We have reached the stage where slides are no longer useful
- Time to switch off slides and get the blackboard dirty